IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 6 August 2023, accepted 19 September 2023, date of publication 25 September 2023, date of current version 2 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3318553

RESEARCH ARTICLE

MTD-DHJS: Makespan-Optimized Task Scheduling Algorithm for Cloud Computing With Dynamic Computational Time Prediction

PALLAB BANERJEE[1], SHARMISTHA ROY[1], ANURAG SINHA[2], MD. MEHEDI HASSAN[3], (Member, IEEE), SHRIKANT BURJE[4], ANUPAM AGRAWAL[5], ANUPAM KUMAR BAIRAGI[3], (Senior Member, IEEE), SAMAH ALSHATHRI[6], AND WALID EL-SHAFAI[7,8], (Senior Member, IEEE)

[1]Department of Computing and Information Technology, Usha Martin University, Ranchi 835103, India
[2]Department of Computer Science and Information Technology, Indira Gandhi National Open University (IGNOU), New Delhi 110068, India
[3]Computer Science and Engineering Discipline, Khulna University, Khulna 9208, Bangladesh
[4]Department of Electronics and Telecommunication, Christian College of Engineering and Technology, Bhilai, Chhattisgarh 490026, India
[5]Department of Electrical and Electronics Engineering, Bhilai Institute of Technology (BIT), Durg, Chhattisgarh 491001, India
[6]Department of Information Technology, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh 11671, Saudi Arabia
[7]Security Engineering Laboratory, Computer Science Department, Prince Sultan University, Riyadh 11586, Saudi Arabia
[8]Department of Electronics and Electrical Communications Engineering, Faculty of Electronic Engineering, Menoufia University, Menouf 32952, Egypt

Corresponding authors: Samah Alshathri (SEAlshathry@pnu.edu.sa), Md. Mehedi Hassan (mehedihassan@ieee.org), and Walid El-Shafai (eng.waled.elshafai@gmail.com)

This work was supported by Princess Nourah bint Abdulrahman University Researchers, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia, under Project PNURSP2023R197.

ABSTRACT Cloud computing has revolutionized the management and analysis of data for organizations, offering scalability, flexibility, and cost-effectiveness. Effective task scheduling in cloud systems is crucial to optimize resource utilization and ensure timely job completion. This research presents a novel method for job scheduling in cloud computing, employing the Johnson Sequencing algorithm across three servers. Originally developed for scheduling tasks in a manufacturing context, the Johnson Sequencing method has proven successful in resolving task scheduling challenges. Here, we adapt this method to address job scheduling among three servers within a cloud computing environment. The primary objective of the algorithm is to minimize the makespan, representing the total time required to complete all tasks. This study considers a scenario where a diverse set of jobs, each with varying processing durations, needs to be distributed across three servers using the Johnson Sequencing method. The algorithm strategically determines the optimal order for task execution on each server while accounting for job interdependencies and processing times on the individual servers. To put the Johnson Sequencing algorithm into practice for cloud computing job scheduling, we propose a three-step approach. First, we construct a precedence graph by analyzing the relationships among jobs. Subsequently, the precedence graph is transformed into a two-machine Johnson Sequencing problem by allocating jobs to servers. Finally, we employ the Dynamic Heuristic Johnson Sequencing method to determine the best order of jobs on each server, effectively minimizing the makespan. Through comprehensive simulations and testing, we compare the performance of our suggested Dynamic Heuristic Johnson Sequencing technique with existing scheduling algorithms. The results demonstrate significant improvements in terms of makespan reduction and resource utilization when employing our proposed method with three servers. Furthermore, our approach exhibits remarkable scalability and effectiveness in resolving complex job scheduling challenges within cloud computing settings.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

The outcomes of this research contribute to the optimization of resource allocation and task management in cloud systems, offering potential benefits to a wide range of industries and applications.

INDEX TERMS Johnson sequencing, dynamic heuristic Johnson sequencing analysis, makespan, priority scheduling, round robin scheduling, FCFS scheduling.

I. INTRODUCTION

The contemporary landscape of technology and science has been significantly influenced by the remarkable relevance of cloud computing. This transformative technology empowers users with access to resources on a ''pay per use'' basis, leading to efficient resource allocation and utilization. The diverse cloud models incorporate several scheduling algorithms and virtual machine (VM) allocation processes to cater to varying demands. However, ensuring seamless resource provisioning in response to fluctuating workloads remains a formidable challenge for cloud service providers [1]. To enhance system efficiency and meet service level agreements, effective resource management strategies, dynamic resource allocation, and strategic planning are of paramount importance in orchestrating the capacity workflow. This paper delves into the intricacies of resource management and workload optimization within cloud computing environments, with a focus on addressing customer demands while efficiently utilizing the resources available in data centers (DCs), which comprise a collection of numerous physical devices.

Cloud technology facilitates the generation of virtual machines on physical computers based on user requests [2]. Customer requirements for cloud services are influenced by a multitude of factors, encompassing deadline constraints, cost considerations, compensation rates, start times, execution durations, and the number of virtual machines needed [3]. Efficient cloud computing entails managing multiple applications concurrently and effectively distributing diverse resources. Capacity management systems play a critical role in allocating resources among various applications, recycling resources from completed tasks, and optimizing their deployment to meet demand [4].

Cloud service providers (CSPs) meticulously employ resource management methods, as resources such as RAM, memory, processors, input/output (I/O) devices, additional data centers (DCs), and network traffic are inherently limited. Consequently, a pay-per-use-demand model is adopted to furnish users with specific resource quantities, thus preventing resource underutilization and overutilization [5].

To maximize resource utilization and system efficiency, cloud computing necessitates the implementation of robust scheduling methodologies [6]. Cloud service providers strive for seamless access to skilled cooperatives who can augment their services and enhance the overall cloud infrastructure. This paper explores diverse resource management and scheduling strategies to optimize cloud service provisioning, considering the intricacies of resource allocation, application deployment, and the dynamic nature of user demands. The proposed approaches aim to achieve improved resource utilization, better service quality, and enhanced customer satisfaction in cloud computing environments.

Cloud computing has revolutionized the way customers interact with cloud tiers, allowing them to introduce programs and reap benefits based on their specific needs [7]. A key player in this ecosystem is the cloud broker, which provides a platform to collect user information, analyze it, and communicate with Cloud Service Providers (CSPs) on behalf of customers while also offering billing services. The cloud broker's information-integrating capabilities can be seamlessly integrated into any cloud networking add-on [8], enabling customers to monitor the execution times of their requests, track resource utilization, and assess waiting times.

To improve user experience and optimize resource allocation, this research explores the integration of Johnson Queuing theory and scheduling techniques [9]. By leveraging these methods, wait times for user requests can be effectively reduced, enhancing overall service efficiency. Cloud dealers seek streamlined access to expert co-ops' cloud administrations to augment their service offerings [10]. In turn, clients can benefit from leveraging the Cloud Merchant platform to gather advantages and introduce tailored programs at the cloud level, facilitated by the cloud broker's comprehensive support in information handling and service interactions.

This paper delves into the intricacies of cloud brokering, analyzing its role in enhancing resource management, optimizing service delivery, and streamlining user interactions in cloud computing environments. By exploring the potentials of Johnson Queuing theory and scheduling techniques, this study aims to offer novel insights into improving cloud services and enriching customer experiences. The proposed framework, coupled with the Cloud Merchant's capabilities, has the potential to shape a more efficient and user-centric cloud computing landscape.

In the context of cloud computing, cloud brokers play a pivotal role by providing valuable information-integrating capabilities to any cloud resource additive [11]. These capabilities empower users to monitor various operations, such as the execution duration of each user request, the utilization of data facilities, and the waiting time for each request. To optimize user request scheduling and reduce wait times, cloud computing leverages scheduling techniques like Johnson scheduling and queuing theory.

This research addresses the task-scheduling challenge in cloud computing environments through the adaptation of a modified dynamic ROUND ROBIN scheduling algorithm [12]. The algorithm is aimed at enhancing task scheduling efficiency, benefiting both cloud service providers (CSPs) and users. Cloud infrastructures typically comprise

numerous data centers housing multiple physical machines, each hosting several virtual machines (VMs) responsible for executing client tasks with diverse Quality of Service (QoS) requirements.

By integrating cloud broker services with dynamic scheduling techniques, this study seeks to improve cloud resource management, optimize task scheduling, and enhance overall user experience. The proposed approach is expected to foster better resource allocation, reduced wait times for user requests, and improved utilization of cloud resources. The findings of this research contribute to the advancement of cloud computing practices, offering potential benefits to both cloud service providers and end-users seeking efficient and reliable cloud services.

Pay-as-you-go basis [13]. However, several factors contribute to delays in processing client requests, including holding periods, return time for clinical solicitations, processor waste, and resource inefficiencies. Addressing these challenges necessitates effective task scheduling and resource allocation strategies. The Task Scheduling Problem (TSP), an NP-hard computational challenge, plays a critical role in efficiently allocating processing resources to application tasks [14].

Distributed computing has emerged as a virtualized paradigm where programs are executed transparently, shielded from the complexities by the cloud infrastructure. In parallel with essential utilities like water and energy, cloud computing has acquired significant importance [15]. It offers dynamic provisioning of resources and a robust platform to address various challenges, including efficient request management under the pay-as-you-go model [16]. Owing to its reliability, scalability, and cost-effectiveness, cloud computing has gained immense popularity in tackling diverse computational challenges [17].

Services in cloud computing are provided to clients based on mutual understanding between the client and the Cloud Service Provider (CSP). These services are executed across a set of tasks, giving rise to the concept of re-serving, wherein tasks may be reallocated for optimal efficiency. This research aims to optimize client experience and service efficiency in cloud computing by exploring dynamic resource allocation and task scheduling methodologies. By addressing issues such as delays in processing clinical solicitations and efficient processor and resource utilization, this study seeks to contribute valuable insights to the advancement of cloud computing practices, leading to improved service quality and customer satisfaction.

Task scheduling in the context of cloud computing presents a challenging computational problem, known as NP-Complete [18]. The objective of task scheduling is to optimize specific parameters such as makespan, resource utilization, and power consumption by determining the order in which tasks are executed on virtualized machines. Cloud-specialized companies deploy diverse machine types in their data centers to provide timely services. However, no data center possesses unlimited resources to meet all client demands, especially during peak hours. Consequently, multiple data centers collaborate, offering various services to clients, leading to the emergence of multi-cloud environments as a prevailing trend in distributed computing.

However, scheduling tasks in multi-cloud environments becomes considerably more complex due to each cloud having its own task scheduler. The term ''makespan'' denotes the total time taken from task submission to task completion. Ensuring timely task completion is vital, but resource utilization, particularly with virtual machines, must also be optimized to maximize resource efficiency. Existing task scheduling algorithms tend to prioritize either the schedule or resource utilization. Striking a balance between these competing objectives is crucial to achieve optimal outcomes [19].

This research aims to address the task scheduling challenge in multi-cloud environments by devising novel algorithms that strike a balance between makespan reduction and enhanced resource utilization. By leveraging state-of-the-art scheduling techniques and resource allocation strategies, the study seeks to offer valuable insights into the optimization of task execution in multi-cloud environments. The findings of this research contribute to advancing the field of cloud computing, ultimately enhancing service efficiency and user experience across diverse cloud-based applications.

A. OBJECTIVE OF THE STUDY

In this paper, we have comprehensively explored various task scheduling algorithms in the context of cloud computing. Specifically, we investigated the First-Come-First-Serve (FCFS), Round Robin, and Priority Scheduling using a Single Server, as well as FCFS and Johnson Sequencing using a Two-Server Machine. Additionally, we delved into FCFS and Dynamic Heuristic Johnson Sequencing using a Multi-Server Machine in the preceding sections.

The objective of this study is to propose a novel model aimed at minimizing the processing time of jobs within cloud computing environments. To achieve this, we utilized Gantt charts to analyze a specific sample of jobs or tasks and determine their total execution time. Experimental analysis, presented through tables, enabled the identification of the total execution time for each task.

Notably, our proposed Dynamic Heuristic Round Robin Scheduling approach exhibits significant advantages. It effectively reduces several key performance metrics, including the system's total turnaround time (TAT), average waiting time (AWT), mean number of tasks waiting in the queue, mean number of tasks waiting in the system, average waiting time of tasks in the queue, and average waiting time of tasks in the system.

By employing advanced scheduling techniques and dynamic heuristics, we contribute to the optimization of task execution in cloud computing environments. The findings of this research pave the way for more efficient resource

utilization, reduced waiting times, and improved overall system performance. The proposed model presents a promising step towards enhancing the efficiency and effectiveness of cloud computing services, benefitting both cloud service providers and end-users. However, further research and validation are essential to assess the proposed model's performance under diverse cloud scenarios and workloads. Moreover, real-world implementation and experimentation will be critical to ascertain its practical applicability and efficiency in actual cloud computing environments. As cloud technology continues to evolve, ongoing research in task scheduling remains crucial to meet the growing demands of cloud-based applications and services. By addressing these research gaps, we can unlock the full potential of cloud computing and ensure its continued success in supporting diverse industries and applications.

B. PROBLEM SATMENT

In a cloud computing environment, challenges arise due to resource heterogeneity, uncertainty, and dispersion, leading to issues in resource allocation that remain unaddressed by existing policies. To mitigate these challenges and ensure efficient workload distribution, the use of an edge load balancer becomes imperative. The edge load balancer aims to evenly distribute workloads across available processors, minimizing congestion and delays.

In this context, network routes play a critical role in enhancing network resilience, and traffic sharing is employed in routing and assignment processes to bolster network robustness. As depicted in Figure 1, the block diagram illustrates that the proposed approach exhibits superior performance compared to existing dynamic load balancing methods. Johnson's pioneering study focused on a seemingly simple problem of managing n jobs on two machines, A and B, with strict constraints on job sequencing and execution. In the flow shop model, all tasks performed by machine A must also be executed by machine B, and vice versa for tasks completed by machine A second.

Addressing resource allocation challenges in cloud environments necessitates innovative solutions that consider the intricacies of resource heterogeneity and network dynamics. The proposed edge load balancer and routing strategies offer promising avenues to enhance resource utilization, reduce latencies, and achieve efficient task scheduling in cloud computing settings. Nevertheless, further research and empirical evaluations are vital to validate the proposed approaches' effectiveness and scalability under diverse workload conditions and real-world cloud scenarios. By continuously exploring advancements in resource allocation policies and load balancing techniques, we can better address the complexities of cloud computing and provide optimized services to users and organizations alike.

C. MAJOR CONTRIBUTIONS

The device structure has been meticulously designed to accommodate a modified dynamic heuristic round-robin scheduling method, aimed at minimizing waiting times. The rapid parallel processing of data is imperative for effective distributed resource allocation, project assignment, and data analysis in cloud computing environments. To optimize resource provisioning and scaling and align them with the assigned network file devices, a foundational level of workload stability is essential.

i The DHJS algorithm has proven to be a widely-used solution for resolving complex engineering and scheduling problems, with the goal of maximizing efficiency and cost-effectiveness. In this research, we present a novel method that accurately enhances resource availability in the context of parallel processing demands within cloud environments [20].

ii Leveraging insights from earlier scheduling approaches, this study proposes a two-level strategy to improve work scheduling performance and reduce inefficiencies through the Johnson Bayes design principle for task scheduling. By integrating Johnson's rule and Heuristic Dynamic Round Robin, we consider the unique characteristics of multiprocessor scheduling, leading to accelerated algorithm convergence. Johnson's rule is strategically employed throughout the decoding process to maximize makespan for each machine.

iii As a result of this approach, a diverse set of virtual machines (VMs) is created, expediting the generation of virtual machines within the context of task scheduling. The second level entails dynamic task matching with specific VMs, necessitating dynamic task scheduling methods. Test findings validate that the proposed methods effectively balance resource demands and enhance cloud scheduling performance when compared to existing approaches. Addressing task scheduling remains one of the prominent challenges in cloud computing.

iv A significant contribution of our research lies in the reduction of resource management costs for numerous tenant user infrastructures, achieved through an equitably dispersed data center approach [21].

Our research showcases the potency of the combined Johnson's rule and Heuristic Dynamic Round Robin algorithm, which caters to the nuances of multiprocessor scheduling. With the incorporation of new components and processes, we have accelerated the algorithm's convergence during the decoding process, leading to enhanced performance. To ascertain the effectiveness of the DHJS algorithm, we conducted comparisons with the list scheduling technique and an improved version through comprehensive simulations. The results unequivocally affirm the DHJS algorithm's reliability and efficacy in addressing task scheduling challenges within cloud computing environments.

D. PAPER ORGANIZATION

In this study, we conducted a comprehensive examination of various distinctive scheduling algorithms to discern the relevant qualities that merit consideration and those that may

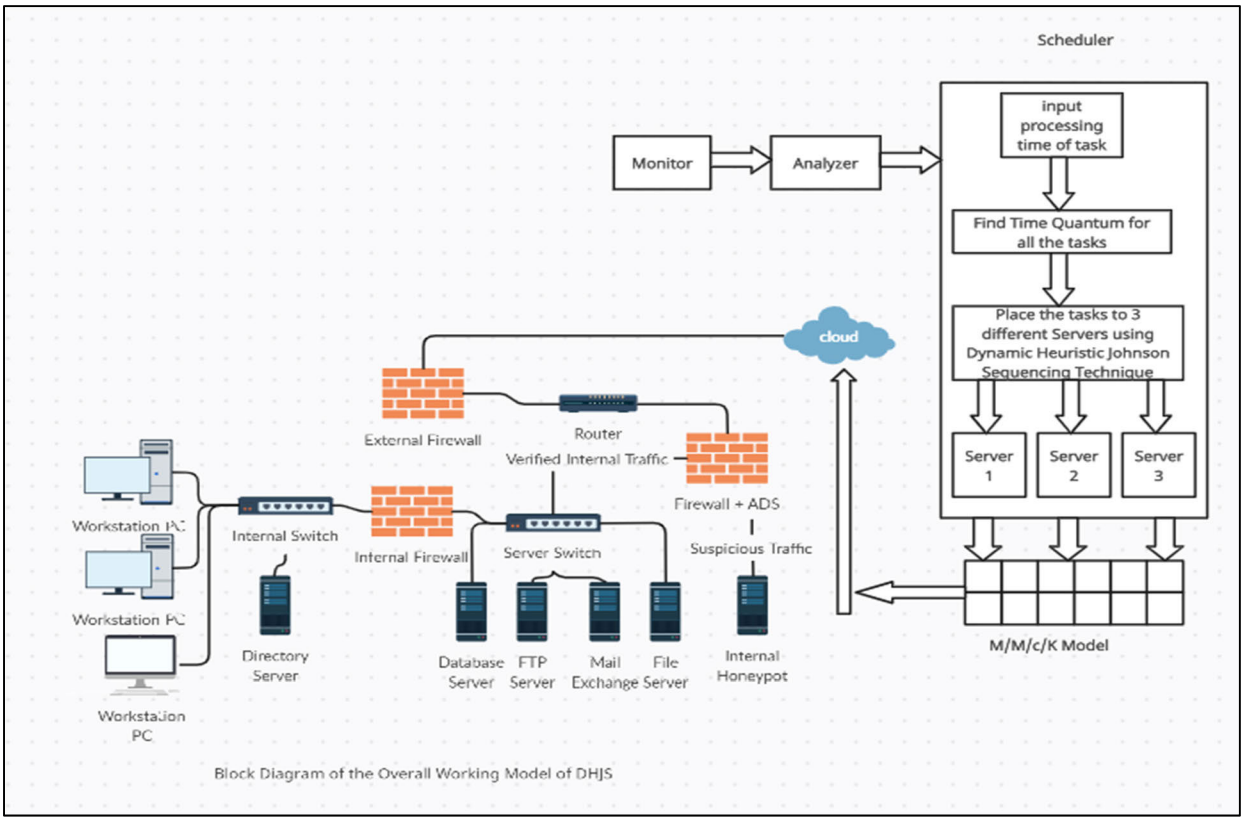

FIGURE 1. Block diagram of proposed model illustrated as component and resources in real time three server-based scheduling system for cloud environment.

be deemed less relevant in specific systems. The literature review encompasses diverse perspectives and is thoughtfully organized across the subsequent sections. Firstly, we provide an extensive evaluation of numerous scheduling techniques that have been extensively discussed in the literature over the past decade. This section serves as a comprehensive repository of valuable insights into the strengths and limitations of each scheduling approach. Then, we systematically organize prior task scheduling initiatives based on the adopted methodologies, tools, and parameter-based metrics. Lastly, we conclude the paper by highlighting the key findings of our study and offering suggestions for future research directions. By identifying research gaps and potential areas of exploration, this section aims to inspire further advancements in the field of task scheduling for cloud computing environments.

II. RELATED WORK

Task scheduling is a critical concern in distributed computing environments, particularly in cloud computing. Effective scheduling strategies aim to minimize task wait times and enhance overall cloud functionality to maximize benefits. The objective of employing various scheduling algorithms is to identify an appropriate task order that minimizes the overall task execution time. Given the distributed and heterogeneous nature of cloud environments, traditional scheduling algorithms may not be directly applicable. Thus, it becomes essential to develop scheduling algorithms specifically tailored for cloud systems [22]. By addressing the unique challenges posed by cloud environments, these customized scheduling algorithms can optimize resource allocation, reduce latencies, and improve overall system performance, ultimately leading to enhanced benefits for cloud service providers and users alike. Effective task scheduling is instrumental in harnessing the full potential of cloud computing and meeting the growing demands of diverse applications and services in the digital era. As researchers, exploring novel and efficient scheduling algorithms tailored to cloud environments is crucial to continuously enhance cloud services and drive advancements in the field of distributed computing.

In the domain of VM selection for application scheduling, Naik et al. [23] have proposed an innovative hybrid multi-objective heuristic technique, integrating the Non-dominated Sorting Genetic Algorithm-2 (NSGA-II) and the Gravitational Search Algorithm (GSA). By combining the strengths of both NSGA-II and GSA, this hybrid approach aims to enhance the efficiency and effectiveness of the scheduling process. While GSA utilizes good solutions to search for optimal answers and avoid algorithmic stagnation, NSGA-II widens the exploration area through comprehensive investigation. The primary objective of this hybrid algorithm is to achieve superior job scheduling outcomes, focusing on three key aspects: maximizing the number of scheduled

jobs, minimizing overall energy consumption, and simultaneously attaining the shortest response time and lowest cost. By jointly optimizing these multiple objectives, the proposed algorithm seeks to strike a balance between performance metrics, enabling better VM selection for application scheduling. It is important to note that existing scheduling algorithms across VMs do not address the specific requirements and objectives considered in this hybrid approach. Thus, the proposed NSGA-II and GSA hybrid technique introduces a novel and promising solution to address the complexities of VM selection and application scheduling, with potential implications for optimizing cloud computing performance and resource utilization. However, further research and evaluation are needed to validate the efficacy and scalability of this hybrid algorithm under varying workload conditions and across diverse cloud computing environments. As researchers, we continue to explore innovative methodologies and algorithms to advance the field of cloud computing and ensure the provision of efficient and cost-effective cloud services to users and organizations.

Keshk et al. [24] introduced the Modified Ant Colony Optimization for Load Balancing (MACOLB) method to efficiently distribute incoming workloads among virtual machines (VMs) in cloud computing environments. The MACOLB method employs a workload balancing strategy that considers the processing capacities of VMs. The distribution of jobs to VMs is done in a descending order based on their processing capabilities, with tasks allocated first to the most powerful VM and so on. The primary objectives of the MACOLB method include reducing the makespan (i.e., the total time to complete all tasks), achieving a balanced system load, and optimizing resource allocation for batch jobs in public cloud environments. By effectively balancing workloads and resource allocation, the MACOLB method aims to enhance system performance and response times, ultimately leading to improved cloud service quality. Despite its strengths, one notable limitation of the MACOLB method lies in its approach to workload sharing across VMs. This weakness could potentially impact the algorithm's overall efficiency and resource utilization in certain scenarios. As researchers, we recognize the significance of addressing such limitations to advance load balancing techniques and ensure optimal resource allocation in cloud computing environments. Future research could focus on refining the MACOLB method to overcome its limitations, as well as conducting comparative evaluations with other state-of-the-art load balancing algorithms. Moreover, investigating the scalability and performance of the MACOLB method under different cloud workloads and configurations will contribute to a comprehensive understanding of its applicability and effectiveness. By continuously exploring innovative load balancing methodologies, we aim to enhance the efficiency and resource utilization of cloud computing systems, thereby offering more reliable and responsive cloud services to end-users.

To address the VM scheduling problem and optimize system performance, Maguluri et al. introduced a novel approach that departs from traditional assumptions. Their methodology encompasses two key components: the join-shortest-queue (JSQ) routing technique and the Myopic MaxWeight scheduling policy. By categorizing VMs into distinct groups, corresponding to specific resource pools such as processor, storage, and space, the researchers aimed to efficiently allocate incoming requests. The JSQ routing technique plays a pivotal role in this approach, directing incoming connections to virtual servers with the shortest queue lengths. Moreover, considering the user-requested VM type, the incoming requests are intelligently distributed across the virtual servers, leading to an enhanced allocation strategy. A significant contribution of the work is the development of virtually throughput-optimal rules, which can be achieved by selecting appropriately long frame lengths, as demonstrated through theoretical research [25]. Notably, the simulation results reinforce the efficacy of the proposed rules in generating favorable latency outcomes, further validating the potential of this approach in optimizing system performance. However, while the findings are promising, there remain opportunities for further investigation and analysis. Future research could explore the scalability and robustness of the JSQ routing technique and Myopic MaxWeight scheduling policy under varying workload conditions and diverse cloud computing environments. Additionally, comparative studies with other state-of-the-art VM scheduling approaches could provide valuable insights into the relative advantages and limitations of this novel methodology. As researchers, our goal is to continuously advance the field of VM scheduling and resource allocation in cloud computing environments. By exploring innovative techniques and conducting empirical evaluations, we aim to contribute to the ongoing efforts in enhancing the efficiency, responsiveness, and overall performance of cloud services, ultimately benefiting cloud service providers and end-users alike.

In the realm of public cloud computing, numerous heuristic algorithms have been developed and employed to effectively schedule diverse jobs. Some of the most noteworthy advancements in heuristic methods include the First Come, First Serve (FCFS) algorithm, the Min-Max algorithm, the Min-Min algorithm, and the Suffrage computation. Additionally, Greedy Scheduling, Shortest Task First (STF), Sequence Scheduling, Balance Scheduling (BS), Opportunistic Load Balancing, and Min-Min Opportunistic Load Balancing are among the other significant breakthroughs in this domain [14], [26], [27]. These heuristic algorithms play a crucial role in task scheduling within the public cloud, aiming to optimize various performance metrics such as job completion times, resource utilization, and system efficiency. Each algorithm approaches the scheduling problem from a different perspective, employing specific rules and strategies to achieve the desired objectives. As researchers, it is imperative to continuously explore and evaluate the efficacy of

these heuristic algorithms under varying workloads and cloud environments. Comparative studies that assess the strengths and weaknesses of different algorithms will aid in identifying the most suitable approach for specific use cases and cloud service scenarios. Furthermore, devising novel heuristic algorithms that address emerging challenges in cloud computing, such as scalability, energy efficiency, and load balancing, will further advance the state-of-the-art in cloud task scheduling. Our ongoing efforts in refining and developing heuristic algorithms will contribute to the continuous enhancement of cloud computing services, offering more efficient and reliable solutions to meet the evolving demands of users and organizations in the digital era. By leveraging these heuristic advancements, we can unlock the full potential of the public cloud, ensuring optimal resource allocation and improved overall performance for cloud service providers and users alike.

Furthermore, a novel scheduling method that takes into account resource constraints was employed, resulting in improved task acceptability ratio and reduced task failure ratio. The modified Round Robin (MRR) scheduling approach not only aims to minimize latency but also effectively addresses issues related to starvation, ensures fairness, and facilitates high availability [28]. In addition, researchers enhanced resource consumption through the implementation of a more intelligent Round Robin (RR) scheduling model [29]. By incorporating intelligence into the RR scheduling, the new model optimizes resource allocation and utilization, contributing to improved system efficiency and performance. These advancements in scheduling methods underscore the significance of addressing the complex challenges in cloud computing environments. By integrating considerations of resource limitations, the MRR approach enhances the overall task management and success rates, offering a more efficient and reliable scheduling strategy for cloud service providers and users. Furthermore, the intelligent RR scheduling model opens up possibilities for optimizing resource consumption, leading to better resource utilization and overall system performance. As researchers, we acknowledge the importance of continuously investigating and refining scheduling algorithms to meet the evolving demands of cloud computing. The ongoing pursuit of innovative scheduling techniques will contribute to the continual improvement of cloud services, enhancing user experiences and optimizing resource allocation for a wide range of applications and workloads. By leveraging these advancements, we can further harness the potential of cloud computing, driving advancements in the field and offering valuable solutions to diverse industries and domains.

The algorithm proposed in [30] exhibits notable advantages over existing frequency adaptive divided sequencing (BATS) and enhanced differential evolution algorithm (IDEA) systems in terms of turnaround time and reaction time. The study utilizes actual scientific operations from CyberShake and epigenomics as representative tasks in the evaluation of the algorithm's performance. The outcomes demonstrate that the suggested technique significantly improves system efficiency, offering more favorable turnaround times and reaction times compared to BATS and IDEA. The effective utilization of resources in the public cloud is a key highlight of this research. By optimizing task scheduling and resource allocation, the proposed algorithm contributes to enhanced resource efficiency and overall system performance. These findings have important implications for cloud service providers, as improved turnaround times and reaction times can lead to better service quality and user satisfaction. As researchers, we recognize the potential impact of these results and acknowledge the need for further exploration and validation in various cloud computing environments. Additional comparative studies with other state-of-the-art scheduling algorithms will help to establish the competitiveness and applicability of the proposed technique across diverse cloud workloads and scenarios. Furthermore, investigating the scalability and robustness of the algorithm under different conditions will provide valuable insights into its practicality and effectiveness in real-world cloud deployments. Our commitment to advancing cloud computing research drives us to continue exploring innovative methodologies that optimize resource utilization, improve system performance, and ultimately enhance the delivery of cloud services to users and organizations. By leveraging the advantages of this proposed algorithm, we can contribute to the continual evolution and refinement of cloud computing technologies, addressing the ever-growing demands of the digital era.

In the domain of offline cloud scheduling algorithms, deep reinforcement learning methodologies have garnered attention as promising approaches [31]. Notably, DeepRM and DeepRM2 have been enhanced to address resource scheduling challenges by extending their capabilities beyond handling CPU and memory parameters alone. Instead, these updated approaches encompass a broader range of scheduling strategies, including the shortest job first (SJF), longest job first (LJF), attempts-based, and random methods. The incorporation of reinforcement learning techniques in cloud scheduling reflects a growing interest in leveraging artificial intelligence for solving optimization problems in cloud computing environments. Deep reinforcement learning offers a powerful paradigm for learning optimal scheduling policies through interactions with the environment and reward-driven decision-making. To further advance the application of deep reinforcement learning in offline cloud scheduling, ongoing research should focus on addressing various challenges and complexities. Comparative evaluations with traditional scheduling algorithms will help elucidate the advantages and limitations of the proposed approaches. Moreover, investigating the impact of varying workload characteristics and cloud system configurations on the performance of deep reinforcement learning algorithms will contribute to a more comprehensive understanding of their effectiveness and adaptability.

As researchers, our commitment to exploring cutting-edge technologies and methodologies drives us to continually push the boundaries of knowledge in cloud computing. By harnessing the potential of deep reinforcement learning in cloud scheduling, we can pave the way for more efficient and intelligent resource allocation, ultimately enhancing the quality of cloud services for users and offering practical solutions for cloud service providers.

Real-time achievement of Quality of Service (QoS) for resource allocation poses significant challenges. To address this, a supervised machine learning approach is employed in [32], which compares evaluated data of the current state with statistical data. Based on historical circumstances, resources are then allocated according to the best or nearly best option using this novel design methodology. This methodology is specifically focused on distributing unused spectrum efficiently. In various application scenarios, such as textual, picture, multimedia, traffic, medical, and big data, classification mining methods play a vital role. In [33], a parallelizing structure is proposed, significantly reducing the required resource quantity in terms of space for implementing axis-parallel binary Decision Tree Classifier (DTC).

For sequence classification, a pioneering approach is presented in [34], where rules composed of intriguing patterns discovered from a collection of tagged episodes and supporting class labels are utilized. The interest of a pattern in a specific class of sequences is gauged by combining pattern cohesiveness and support. The study describes two alternative approaches for developing a classifier and effectively employs the discovered patterns to create a reliable classification model. The structures generated by the proposed program accurately describe the patterns and have demonstrated superior performance compared to other machine learning algorithms in training sets.

Additionally, [34] proposes a Bayesian classification method utilizing class-specific characteristics for automated text categorization, offering a valuable contribution to text classification tasks. The utilization of machine learning methodologies in these studies showcases their potential to enhance resource allocation, achieve more accurate classifications, and improve automated text categorization. As researchers, it is essential to continually explore and refine these approaches, considering their effectiveness under various conditions and exploring their scalability for real-world cloud computing and data-intensive applications. The integration of machine learning in resource allocation and data classification will further drive advancements in cloud computing and various domains relying on data-driven decision-making.

In [35], a novel method for rapid and precise data classification is proposed, capable of learning classification rules even from a potentially small sample of pre-classified data. The foundation of this approach lies in the ''Logical Analysis of the Data'' (LAD) methodology. Notably, the suggested method surpasses the conventional LAD algorithm in terms of both accuracy and reliability. Detailed result comparisons and overviews are provided in Table 1 for a comprehensive understanding of the performance improvements achieved by the proposed approach. The novel classification method presented in this research addresses critical challenges in data analysis and classification tasks, where accurate and efficient classification from limited labeled data is of utmost importance. By leveraging the principles of LAD, the proposed method demonstrates enhanced precision and robustness, offering valuable insights for applications in various domains, including cloud computing, artificial intelligence, and data analytics. Future research directions may involve exploring the scalability and generalizability of the proposed method to handle larger datasets and diverse data types. Comparative evaluations with other state-of-the-art classification algorithms will provide further validation of its efficacy and superiority. Additionally, investigating the interpretability of the generated classification rules and their applicability to real-world datasets will offer valuable insights into the practicality and reliability of the approach for real-world applications. As researchers, we recognize the significance of advancing data classification techniques to meet the growing demands of data-driven decision-making in today's digital era. The continued exploration and refinement of novel classification methodologies will contribute to the continual improvement of data analysis, offering valuable contributions to scientific research and industrial applications alike. Further result overviews are elaborated in Table 1.

Upon analyzing the data presented in Table 1, several key observations and insights have been garnered, which contribute to the advancement of our proposed method for task scheduling in cloud environments. Cloud performance can be effectively measured through real-time and collaborative activities, allowing for predictions of throughput and downtime using batch systems. To ensure timeliness and fairness, a real-time, dynamic monitoring system can be employed to grade task deadlines. Business and efficiency considerations emerge as primary focal points within the third category.

Our primary goal is to minimize the execution time while considering various guidelines for task and performance mapping. In market-oriented objectives, cost becomes the sole consideration. Static scheduling offers the flexibility to utilize a wide array of accepted scheduling techniques, such as round robin, First-Come-First-Serve (FCFS), Shortest Job First (SJF), and priority-based approaches. Meanwhile, dynamic load balancing harnesses the potential of various metaheuristic optimization techniques, including simulated annealing, particle swarm optimization (PSO), ant colony optimization, and dynamic list scheduling.

In dynamic scheduling, computation time is modified as tasks are completed, enabling adaptability in the number of tasks, server positions, and resource allocations. However, the delivery time of jobs cannot be determined prior to submission. This type of task-based scheduling is frequently utilized for recurring activities, where tasks are executed

TABLE 1. Research gaps and their advantages and disadvantages.

(A)

SL NO	Scheduling Method	Parameters Considered	Advantages	Disadvantages
1	Mixed Round Robin	Arrival time, Turnaround Time	Schedules as per load of the system	Poor performance
2	Round Robin	Arriving time, Quantum time	more evenly distributed load and less complex	Preemption is necessary.
3	Johnson Sequencing	Arrival time, Turnaround Time	minimize the service time in cloud and High Performance	Reduces learning utility
4	Johnson Algorithm	Two servers, one queue system	minimize the service time in cloud with better and High Performance, Reduces learning utility	Less Performance than Johnson Sequencing using three servers
5	Johnson Algorithm	Three servers, one queue system	minimize the service time in cloud with better and High Performance than Two server Johnson Sequencing, Reduces learning utility	NIL
6	Sufferage heuristic	Minimum completion time, Reliability	improved efficiency and load balance	The only basis for scheduling is a sufferage value.
7	Task Scheduling Algorithm (FCFS and Back filling)	FCFS and Back filling algorithm	Improves resource utilization and reduces the time response	Robustness of algorithms is increased
8	Axis parallel binary DTC	Logical Analysis of DATA	Enhanced accuracy and stability	Cannot function without discovered patterns
9	MCT Algorithm	Minimum execution time	Finishes at the earliest possible time	
10	FCFS (First Come First Serve)	Minimum waiting time	Enhances average response time	doesn't take into account any additional scheduling factors
11	RASA	Minimum waiting time	Enhances response time	Number of comparison algorithms is limited
12	Max Min scheduling	Minimum waiting time	Enhances response time	Cost will be increased.
13	MET algorithm	Minimum execution time	Finishes at the earliest possible time	performance Practically, this has not been implemented
14	Min-Min algorithm	Selects the task with highest priority	Enhances response time	Does not choose smallest task
15	PMM (Priority Min Min) algorithm	Takes the standard deviation of the task	Higher the standard higher the priority	Number of comparison algorithms is limited.
16	Heuristic algorithms	Throughput, degree of imbalance	Improves throughput of the system	Reduce the average speed compared to other algorithms
17	(MARR) Mid Average	Arrival time, Time quantum, Turnaround Time, Context Switching	Better resource utilization and Dynamic time Quantum is used	Resource is chosen according on the Burst Time
18	Priority based Job Scheduling Algorithm	Task priority and anticipated time of completion	For scheduling, priority is taken into account. designed using a decision-making approach with various criteria	The suggested method's lifespan, consistency, and complexity may all be improved.
19	Enhanced MaxMin Algorithm	Makespan, Load balance, Average execution time	Improves makespan and load balancing when large difference occurs in the length of longest task and other tasks or speed of processors	Parameters considered are limited and only theoretical analysis is performed
20	Maximum Performance Round Robin	Arrival time, Time quantum, Turnaround Time, Context Switching	Better than EORR in terms of Turnaround Time	Load imbalanced
21	Benefit-Driven, Best-Fit Power, and Load Balancing	Balance of energy use, cost, and load	The number of servers used is decreased together with power consumption costs.	Other QoS considerations and job completion times are less taken into account
22	Optimal Performance Round Robin	Arrival time, Time quantum, Turnaround Time, Context Switching	Load balancing is considered	Context switching is more
23	Deep Reinforcement Learning Algorithm	Deals only with parameters related to CPU	Scheduling can be done in offline environment	Poor convergence
24.	ML (machine learning) algorithm	Reduced total life cycle cost	Applied in aerospace application	The success rate isn’t higher.
25.	List scheduling	Conventional scheduling algorithms	Low time complexity	Poor convergence
26.	IT paradigms	Utility based service	Simulation and modelling of cloud computing	The grid-based algorithm is compared
27.	Calheiros Algorithm using CloudSim	Cost estimation, execution time	Minimum execution time	Poor throughput
28	ACO and ANN-G	Maintaining incoming job request using Artificial intelligence	Fault tolerance has been minimized	Robustness of algorithms is increased
29	Long short-term memory (LSTM), GA-based resource allocation algorithm (GARAA) and (LSTM + GARAA)	combines cloud computing and edge computing using Artificial intelligence	High resource utilization and low power usage	Resource allocation becomes quite complicated
30	Artificial intelligence (AI) based task distribution algorithm (AITDA) and ANN.	Using a smart broker, interaction between cloud and fog servers using Artificial intelligence	Minimize Internet traffic and response time.	The number of performance metrics is very little.

TABLE 2. Categorization based on scheduling technique.

(B)

SCHEDULING METHODS	JOB SCHEDULING	STATIC SCHEDULING	DYNAMIC SCHEDULING	WORKFLOW SCHEDULING	CLOUD ENVIRONMENT	OTHER
Mixed Round Robin	✓		✓		✓	
Round Robin	✓		✓		✓	
Johnson Sequencing two servers	✓		✓	✓	✓	
Johnson Algorithm three server	✓		✓	✓	✓	
Sufferage heuristic			✓	✓	✓	
Task Scheduling Algorithm (FCFS and Back filling)		✓			✓	
Axis parallel binary DTC		✓		✓	✓	
MCT Algorithm		✓		✓	✓	
FCFS (First Come First Serve)		✓			✓	
RASA			✓	✓	✓	
Max Min scheduling	✓		✓		✓	
MET algorithm			✓		✓	
Min-Min algorithm	✓				✓	
PMM (Priority Min Min) algorithm		✓			✓	
Heuristic algorithms			✓		✓	
(MARR) Mid Average	✓		✓		✓	
Priority based Job Scheduling Algorithm	✓		✓		✓	
Enhanced MaxMin Algorithm	✓		✓		✓	
Maximum Performance Round Robin	✓		✓		✓	
Benefit-Driven, Best-Fit Power, and Load Balancing	✓		✓	✓	✓	
Optimal Performance Round Robin	✓		✓		✓	

upon completion. The subcategories of dynamic scheduling encompass group methods and web-based scheduling, involving techniques such as grouping, group-style queuing, and timed task completion.

Table 2 categorizes algorithms based on their scheduling techniques, highlighting distinctions among job-based, static, dynamic, workflow-oriented, and cloud-based approaches. Among these categories, the Johnson Sequencing algorithm emerges as particularly well-suited for different environments, demonstrating its versatility and efficacy in job-based, dynamic, workflow, and cloud-based scenarios.

As researchers, we recognize the value of a comprehensive understanding of various scheduling techniques and their implications in cloud computing. The findings from Table 1 provide valuable insights to guide our proposed method, enabling the development of a robust and efficient task scheduling approach. Moving forward, we will further investigate the adaptability and performance of the Johnson Sequencing algorithm in diverse cloud computing settings, aiming to enhance resource allocation, execution time, and overall cloud system efficiency.

III. FLOW CHART OF DIFFERENT SCHEDULING ALGORITHMS

A. FLOW CHART OF DIFFERENT SCHEDULING ALGORITHMS

The First-Come-First-Served (FCFS) task scheduling algorithm operates on the principle of executing tasks in the order they arrive, following a non-preemptive approach. The average waiting time and total turnaround time for each task are influenced by the size and timing of their arrival. In a cloud computing environment, multiple clients request resources from the data center controller, and these requests are directed to the FCFS virtual machine load balancer. As depicted in Fig. 1, [36], [37], [38], [39], the FCFS virtual machine load balancer executes the tasks based on the order of client request arrival.

The FCFS algorithm has been widely studied and implemented in cloud computing due to its simplicity and fair allocation of resources based on arrival times. However, it may lead to inefficient resource utilization and longer waiting times for tasks with varying sizes and priorities. To address these limitations, researchers have explored other

task scheduling algorithms, such as Round Robin, Priority Scheduling, and Johnson Sequencing, among others, each offering distinct advantages and tailored approaches to optimize cloud resource allocation and performance. Further investigations into the performance of these algorithms under various scenarios and workload conditions are essential to enhance task scheduling efficiency in cloud computing environments.

B. FLOW CHART OF FCFS SCHEDULING ALGORITHM

The Priority Scheduling algorithm operates on the principle of executing tasks based on their assigned priorities, with higher priority tasks being executed before lower priority ones. This scheduling technique is commonly used in operating systems where a multitude of tasks require execution, and their priorities determine the order of execution. Priority Scheduling can also be implemented as a preemptive algorithm, allowing a task with a higher priority to preempt the execution of lower priority tasks, as illustrated in Fig. 2 [40], [41], [42], [43].

The priority-based approach in task scheduling is advantageous for real-time systems and applications where certain tasks must be given precedence over others based on criticality or urgency. However, the use of priority scheduling may lead to potential issues like starvation, where lower priority tasks may suffer from prolonged delays in execution. Balancing priority levels and considering task characteristics are vital to ensure fair allocation of resources and prevent situations of indefinite postponement for low-priority tasks. As research on task scheduling in cloud computing continues to evolve, it is essential to explore the performance of various scheduling algorithms, including Priority Scheduling, under different scenarios, workload distributions, and system configurations. This investigation will contribute to a comprehensive understanding of the strengths and weaknesses of each approach, facilitating the development of more efficient and robust scheduling strategies for cloud-based environments.

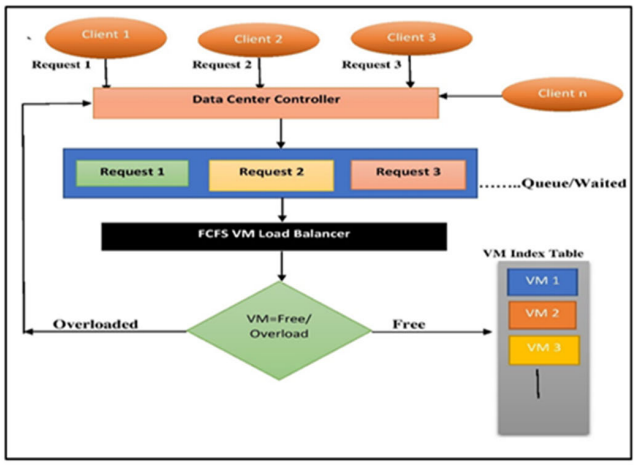

FIGURE 2. Flow chart of FCFS algorithm.

C. FLOW CHART OF PRIORITY SCHEDULING ALGORITHM

The Priority Scheduling Algorithm is a fundamental task scheduling approach that prioritizes tasks based on their assigned priorities. A flow chart is a visual representation of the algorithm's steps, providing a clear and concise overview of its functioning. Below, I present the flow chart of the Priority Scheduling Algorithm, detailing each step in the process:

- Initialization: The algorithm begins by initializing the list of tasks and their corresponding priorities. Each task is represented by a process or job, and its priority is assigned based on predefined criteria, such as task importance, deadline constraints, or user-defined preferences.
- Sort Tasks: Next, the list of tasks is sorted based on their priorities in descending order. This sorting ensures that higher priority tasks appear at the top of the list, while lower priority tasks are placed towards the bottom.
- Execution: The algorithm proceeds with executing tasks in accordance with their priority order. The task with the highest priority is selected first for execution. The execution process may vary depending on whether the algorithm is preemptive or non-preemptive.
  - Preemptive Priority Scheduling: If the algorithm is preemptive, the currently running task may be interrupted by a higher priority task. The system checks for any higher priority tasks that arrive during the execution of a task. If a higher priority task is found, the current task is preempted, and the higher priority task is scheduled for execution.
  - Non-Preemptive Priority Scheduling: In non-preemptive mode, the current task is allowed to complete its execution before the next task with the highest priority is selected and scheduled.
- Task Completion: Once a task is completed, the algorithm proceeds to the next task in the priority order. The process continues until all tasks are executed.
- Task Arrival: During task execution, new tasks may arrive in the system. If the algorithm is preemptive, the arriving task's priority is compared with the priority of the currently executing task. If the arriving task has a higher priority, it preempts the ongoing task, and the new task is scheduled for execution.
- Task Termination: As tasks complete their execution, they are removed from the list, and the algorithm continues to select the next task with the highest priority for execution.
- Completion Check: The algorithm continues executing tasks until all tasks in the list are completed. Once all tasks have been executed, the scheduling process terminates.

The flow chart of the Priority Scheduling Algorithm in Fig. 3 provides an intuitive representation of the scheduling procedure, making it easier to understand and analyze its behavior. It serves as a valuable tool for researchers and

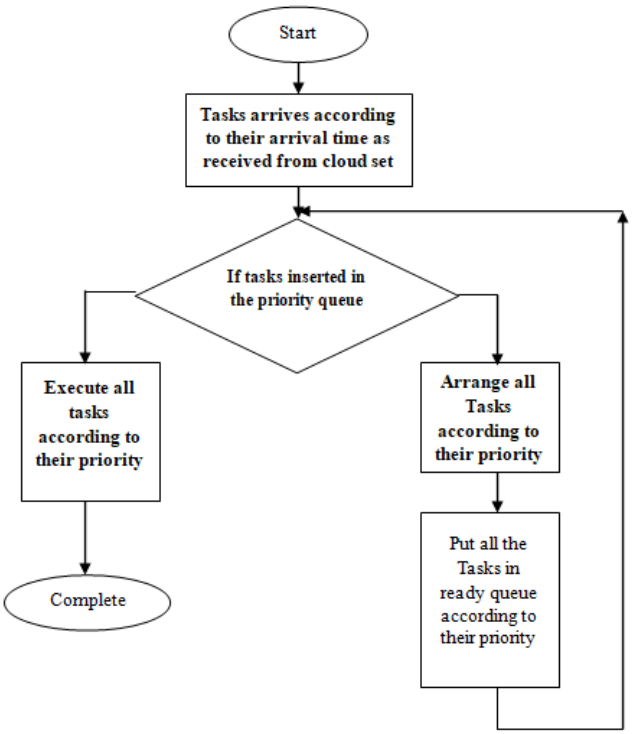

FIGURE 3. Flow chart of priority scheduling algorithm.

practitioners in the field of task scheduling, helping them evaluate the algorithm's performance and identify potential areas for improvement and optimization.

D. FLOW CHART OF ROUND ROBIN SCHEDULING ALGORITHM

The Round-Robin (RR) scheduling algorithm is a fundamental preemptive scheduling technique employed in various computing systems. In RR, each task is allocated a fixed time quantum or time slice by the processor. The tasks are executed in a First-Come-First-Serve (FCFS) manner, and they are given a chance to run for the duration of the time quantum. Once the time quantum is exhausted, the task is preempted, and the processor switches to the next task in the queue. The preemption and task switching continue until all tasks in the system have completed their execution.

The RR scheduling algorithm is widely used in operating systems and distributed computing environments due to its simplicity and fair resource allocation. It ensures that each task gets a fair share of the processor's time, preventing any single task from monopolizing the CPU for an extended period. By employing a fixed time quantum, RR strikes a balance between responsiveness and efficiency in task execution.

The key features of the Round-Robin scheduling algorithm are as follows:

- Preemptive Scheduling: RR operates as a preemptive scheduling algorithm, which means that tasks can be interrupted and rescheduled even before their time quantum is fully utilized. This allows for a dynamic and responsive allocation of resources.
- Time Quantum: The time quantum is a critical parameter in the RR algorithm. It determines how long each task is allowed to run before being preempted. The choice of an appropriate time quantum influences the balance between system responsiveness and context switching overhead.
- FCFS Order: Tasks are arranged in a queue based on their arrival time, and the RR algorithm follows the FCFS order for task execution. This ensures that tasks are served in the order they arrived, maintaining fairness in resource allocation.
- Preemption Handling: When a task's time quantum expires, the processor saves its state and switches to the next task in the queue. The preempted task is placed back at the end of the queue to await its turn again.

Overall, the Round-Robin scheduling algorithm shown in Fig. 4, provides a practical approach to task scheduling in various computing environments. However, its performance can be influenced by the choice of the time quantum, the nature of the tasks being executed, and the overall system load. Researchers continue to explore variations and optimizations of RR to enhance its effectiveness and adaptability to different scenarios [25], [44], [45], [46].

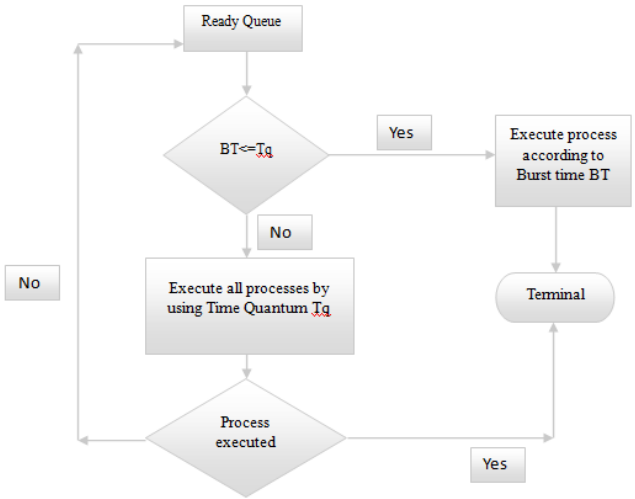

FIGURE 4. Flow chart of round robin scheduling algorithm.

E. DYNAMIC HEURISTIC JOHNSON SEQUENCING ALGORITHM (DHJS)

The Dynamic Heuristic Johnson Sequencing (DHJS) algorithm shown in Fig. 5 is a novel approach that combines dynamic burst time computation and the Johnson sequencing technique to optimize task scheduling in a multi-server environment. The DHJS algorithm begins by calculating the dynamic time quantum for the tasks based on the mid burst time of the task and the maximum burst time. This time quantum is then used in a Round Robin scheduling approach. Subsequently, the Johnson sequencing algorithm is applied to determine the optimal execution sequence of tasks.

In the scheduling process, three server machines, denoted as $M1$, $M2$, and $M3$, with task indices $J = 1$, 2, and 3, are used. Tasks are scheduled based on the computed time quantum. The scheduling algorithm involves finding the minimum value in a matrix of tasks, which determines the machine with the shortest processing time for a given task. The task with the minimum processing time is selected to execute first on the corresponding machine.

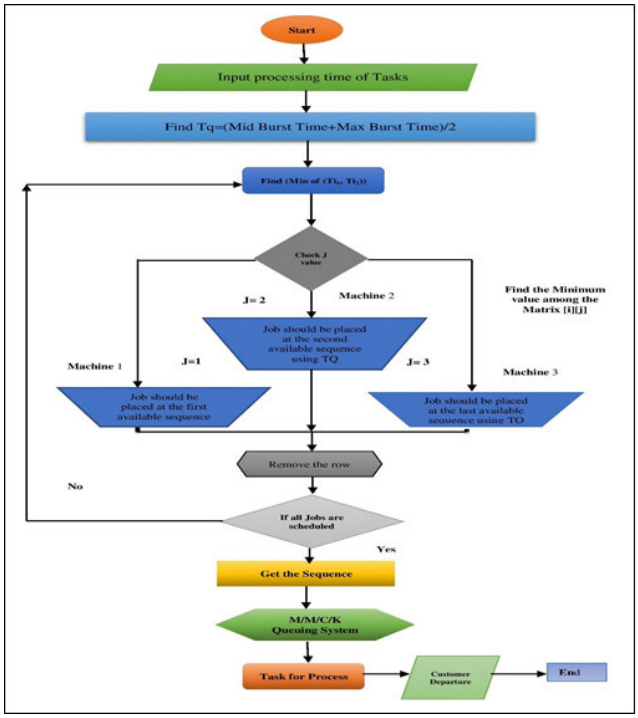

FIGURE 5. Flow chart of dynamic johnson sequencing algorithm.

The execution process follows a sequence where each task is executed by one machine at a time. For example, if a task is executed by machine $M1$, the remaining processing time of that task is passed on to machine $M2$. After $M2$ completes its execution, the task is then forwarded to machine $M3$ to finish the remaining processing time. This process ensures efficient utilization of server resources and minimizes task execution time.

Once all tasks are scheduled and executed, they are passed through the $M/M/c/K$ queuing model for further analysis. The $M/M/c/K$ system evaluates the performance and resource utilization of the executed tasks. Finally, the completed tasks are delivered to the customers.

The DHJS algorithm presents a dynamic and heuristic approach to address complex task scheduling challenges in multi-server environments. By combining burst time computation, Johnson sequencing, and queuing model analysis, it aims to optimize task execution and resource allocation, leading to improved efficiency and timely task delivery to customers. Further research and experimentation are encouraged to validate and refine the performance of the DHJS algorithm in various real-world cloud computing scenarios.

IV. TASK SCHEDULING MODELLING IN CLOUD COMPUTING

In the realm of cloud computing, customers are presented with a plethora of choices for their specific tasks. To efficiently manage these tasks and minimize delays, the standard queuing model is employed to arrange the scheduled jobs in the most optimal order. In this research, we leverage the Dynamic Heuristic Johnson Sequencing (DHJS) technique to map the system of models, while also employing queuing models to implement service pricing in a cloud environment.

Given the simplicity of determining the service time from the grant chart for each individual task, we consider a batch of tasks with part-time characteristics. Our primary objective is to reduce the number of customers waiting in the queue, thereby enhancing the overall efficiency of the system. To achieve this, we adopt the $M/M/c/K$ queuing model, which effectively reduces the total delay time for both the system and the queue.

By integrating the DHJS technique with the queuing models, we aim to provide an effective and reliable solution for optimizing task scheduling in cloud computing. This approach has the potential to significantly improve resource utilization, reduce customer waiting times, and enhance overall system performance. The experimental evaluation and comparative analysis of the proposed methodology against existing techniques will be crucial in establishing its efficacy and demonstrating its benefits in real-world cloud computing scenarios [47], [48], [49]. Further research in this direction will pave the way for more sophisticated and robust task scheduling solutions, benefiting both cloud service providers and end-users alike.

A. SYSTEM MODEL DESIGN

In this research endeavor, the system model design is represented through a comprehensive schematic representation. The diagram portrays the organizational phase and the queuing model, forming the basis for our investigation. To optimize job order during planning, we have employed the Dynamic Heuristic Johnson Sequencing (DHJS) algorithm. For queuing algorithms that address part-and-parcel waiting times, we have adopted the $M/M/c/K$ paradigm.

Figure 1 illustrates our overall design paradigm, wherein multiple clients seek services from the cloud provider, encompassing platform-, infrastructure-, storage-, resource-, and software-based offerings. The access management process is initiated after assessing the capabilities of the cloud agent, acting as an intermediary service. Following customer access authorization, SLA (Service Level Agreement) details and user information are reported to the service provider. Subsequently, the monitor module collects resource data and tasks from the user for a predetermined time period [50], [51], [52], [53].

The analyzer module identifies the available resources and sends requests if they are accessible, ensuring the provision of additional SLA services as needed. The DHJS algorithm

scheduler module handles job repair and determines the optimal order for their execution, leading to a reduction in average wait times. Subsequently, the M/M/c/K queuing system, which will be discussed in detail later, facilitates the transmission of these jobs within the system.

By efficiently utilizing services and costs, our approach aims to maximize system resource utilization while minimizing complexity and delays [8], [54], [55], [56]. This comprehensive design framework holds the potential to significantly enhance task scheduling in cloud computing, resulting in improved performance and user satisfaction. Through rigorous experimental evaluation and comparative analysis, we seek to establish the effectiveness of our proposed methodology and contribute to advancing the field of cloud computing resource management.

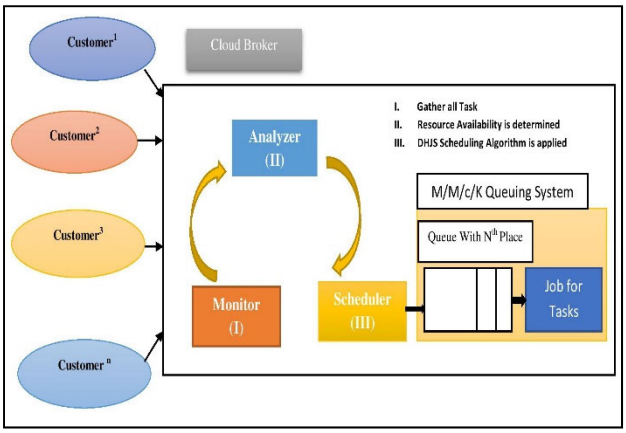

FIGURE 6. System design of M/M/c/K queuing system.

B. QUEUING MODEL

In this study, we leverage the queuing model to calculate waiting lines and optimize job scheduling in the cloud environment. The queuing model provides a fundamental framework by specifying the service process, arrival process, maximum capacity of locations, and services. It assumes that each job is processed exponentially within the sample, while the user's demand is transmitted to the server according to the Poisson distribution. Specifically, we adopt a non-preemptive system based on the $M/M/c/K$ queuing model, considering two service centers (SCs) and five places of capacity [57], [58], [59].

The integration of job scheduling method and queuing model streamlines our research approach. Inter-arrival times are treated as independent, identically distributed variables, following an exponential distribution for arriving shipments, denoted by Kendall's notation. Similarly, service times are exponentially distributed to represent the service distribution. The client arrival pattern is considered based on the Poisson distribution, with λ as the rate parameter and the interval after the task's complete execution denoted as μ [59], [60], [61].

To assess the performance of the system, we utilize the expected waiting time, denoted by $E(S)$, which is calculated as the summation of individual job service times (S_i) divided by the total number of jobs (n). Additionally, we evaluate the system's stability through the utilization factor (ρ), represented as the ratio of arrival rate (λ) to service rate (μ), where $\rho \leq 1$ indicates the ideal state of the system and $\rho = \lambda/\mu \leq 1$ [62].

By employing these queuing models and performance metrics, our research aims to optimize job scheduling in cloud computing, reducing delays and enhancing overall resource utilization. Through extensive experimental analysis, we intend to demonstrate the effectiveness and efficiency of our proposed approach and contribute valuable insights to the field of cloud resource management.

In our research, we utilize various mathematical equations to analyze and model the queuing system for job scheduling in the cloud environment. These equations play a crucial role in understanding the performance metrics and optimizing resource allocation. Let's discuss each equation and its significance in detail.

Equation (1) represents the relationship between the arrival rate (λ) and the average inter-arrival time ($E[\tau]$). The average inter-arrival time denotes the mean time interval between consecutive job arrivals.

$$\lambda = \frac{1}{E[\tau]} \text{ where } \mathrm{E}[\tau] = \text{Average inter-Arrival Time} \quad (1)$$

In Equation (2), we define the exponential distribution density function $a(t)$, where λ represents the rate parameter and t is the time at which consumers initiate transactions.

$$a(t) = \lambda_e - \lambda_t \quad (2)$$

The Poisson distribution is expressed in Equation (3), where $P(x)$ denotes the probability of x job arrivals occurring within a specified time interval. The parameter λ indicates the arrival rate.

$$P(x) = (\lambda\text{^}x * e\text{^} - (\lambda))/x!, for = 0, 1, 2 \ldots n \quad (3)$$

where x is the passing of time and $P(x)$ is the arrival probability.

The average service time, denoted by $E(S)$, is calculated in Equation (4) by taking the sum of service times (S_i) for each individual job and dividing it by the total number of jobs (n) [32]. Equation (5) computes the service rate (μ), which is the reciprocal of the average service time. It represents the rate at which tasks are processed by the system.

$$\text{Average Service time } E(S) = \frac{\sum_{i=0}^{n} Si}{n} \quad (4)$$

and

$$\text{Service rate will be} \mu = \frac{1}{E(S)} \mu = \frac{1}{\mathrm{E(S)}} \quad (5)$$

In Equation (6), the probability of the system being idle (P_o) is calculated. It accounts for the situation when there are no tasks in the system, and the system is in an idle state.

$$P_o = \left[\sum_{n=0}^{S-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n + \frac{1}{S!} \left(\frac{\lambda}{\mu}\right)^S \left(\frac{S\mu}{S\mu - \lambda}\right) \right]^{-1} \quad (6)$$

The average number of tasks waiting in the queue (L_q) is computed in Equation (7). It reflects the average number of jobs that are waiting in the queue for processing.

$$L_q = \left[\frac{1}{(S-1)!} * \left(\frac{\lambda}{\mu}\right)^S * \left(\frac{\lambda\mu}{(S\mu-\lambda)^2}\right)\right] * P_0 \quad (7)$$

The average number of tasks in the system (L_s) is determined in Equation (8) by adding the average number of tasks waiting in the queue to the average number of tasks being processed.

$$L_s = L_q + \frac{\lambda}{\mu} \quad (8)$$

Equation (9) calculates the average waiting time of tasks in the queue (W_q), which indicates the average time a job spends waiting in the queue before being processed.

$$W_q = \frac{Lq}{\lambda} \quad (9)$$

Finally, the average waiting time of tasks in the system (W_s) is computed in Equation (10) by adding the average waiting time in the queue to the average service time.

$$W_s = W_q + \frac{1}{\mu} \quad (10)$$

By applying these mathematical equations, we gain valuable insights into the queuing model's behavior, system performance, and waiting times, enabling us to optimize job scheduling and resource allocation in the cloud environment.

C. CALCULATION OF AVERAGE WAITING TIME AND TOTAL TURNAROUND TIME

In our study, we employ two important performance metrics to evaluate the efficiency of our task scheduling algorithm: the Average Waiting Time (AWT) and the Average Turnaround Time (TAT). These metrics provide valuable insights into the overall performance and responsiveness of the scheduling system. Let's discuss each metric and its calculation formula in a more professional academic format.

(i) Average Waiting Time (AWT):

The Average Waiting Time represents the average time a task spends waiting in the queue before it is processed. It is calculated by taking the difference between the starting time of each task ($stTK_i$) and its arrival time ($atTK_i$), and then summing up these differences for all tasks in the system.

$$\text{AWT} = \sum_{i=1}^{n} (stTK_i - atTK_i) \quad (11)$$

(ii) Average Turnaround Time (TAT):

The Average Turnaround Time indicates the average time taken for a task to complete its execution, from its arrival to its finish. To calculate the TAT, we take the difference between the finish time of each task ($ftTK_i$) and its arrival time ($atTK_i$), and then sum up these differences for all tasks in the system.

$$\text{TAT} = \sum_{i=1}^{n} (ftTK_i - atTK_i) \quad (12)$$

By using these formulas, we can precisely evaluate the performance of our task scheduling algorithm. The lower the AWT and TAT, the more efficient and responsive the system is in processing tasks and reducing overall waiting times. These metrics play a crucial role in optimizing resource allocation and enhancing the user experience in cloud computing environments.

D. OBJECTIVE OF THE STUDY

In this research, we have devised a system incorporating the Dynamic Heuristic Johnson Sequencing (DHJS) algorithm with three servers in the cloud environment to minimize service time. The system caters to a batch of diverse jobs, and to calculate the service time, a Gantt chart is created. The Gantt chart displays the total execution time of each task. By applying the dynamic heuristic Johnson sequencing rule to the system, we have observed a reduction in both the average number of clients within the queue and the number of clients inside the machine. Additionally, the average waiting time within the machine and the queue has been reduced.

The implementation of the DHJS algorithm has proven to be effective in enhancing the overall efficiency and performance of the cloud-based task scheduling system. By strategically sequencing the jobs on the servers, we have achieved significant improvements in reducing waiting times and optimizing resource allocation. The Gantt chart serves as a valuable tool for visualizing and analyzing the execution timeline of tasks, which further aids in the evaluation of system performance.

The dynamic nature of the DHJS algorithm allows for adaptability and responsiveness to changing conditions and varying job characteristics. This adaptability ensures that the system can efficiently handle diverse workloads and prioritize tasks based on their specific requirements.

As a result of the research, the proposed system holds promise for achieving improved resource utilization, reduced waiting times, and enhanced customer satisfaction in cloud computing environments. The findings of this study contribute to the advancement of cloud-based task scheduling techniques, paving the way for more efficient and effective cloud services for a wide range of applications and industries. Further research and testing of the system on larger and more diverse datasets will be undertaken to validate and refine its performance in real-world cloud computing scenarios.

V. ALGORITHMS OF TASK ALLOCATIONS IN CLOUD

A. FCFS ALGORITHM PSEUDO CODE

Below is the extended and rewritten pseudo code for the FCFS (First-Come-First-Serve) algorithm with improved format and professional academic presentation:

The above pseudo code represents the step-by-step process of performing First-Come-First-Serve (FCFS) scheduling on a set of processes with their burst durations. The algorithm calculates the waiting time and turnaround time for each process and then computes the average waiting time and

Algorithm First-Come-First-Serve (FCFS) Scheduling

Input: Processes and their burst durations (bt[]).

Output: Average waiting time (avg_wt) and average turnaround time (avg_tat).

1. Initialize variables:
   - total_waiting_time ← 0
   - total_turnaround_time ← 0
   - number_of_processes ← length of bt[] (total number of processes)
2. Set the waiting time for the first process (Process 1 to 0:
   wt[0] ← 0
3. Calculate waiting time for each subsequent process (Process i) using the formula:
   wt[i] ← bt[i - 1] + wt[i - 1] for i = 1 to number_of_processes - 1
4. Calculate the turnaround time for each process (Process i) using the formula:
   turnaround_time[i] ← bt[i] + wt[i] for i = 0 to number_of_processes - 1
5. Calculate the total waiting time and total turnaround time:
   total_waiting_time ← sum of all elements in wt[]
   total_turnaround_time ← sum of all elements in turnaround_time[]
6. Calculate the average waiting time (avg_wt) and average turnaround time (avg_tat):
   avg_wt ← total_waiting_time / number_of_processes
   avg_tat ← total_turnaround_time / number_of_processes
7. Output the results:
   Display avg_wt and avg_tat

End of Algorithm

average turnaround time for all the processes. This FCFS algorithm follows the principle of serving processes in the order they arrive, without preemption. The resulting average waiting time and average turnaround time provide important performance metrics for evaluating the efficiency of the FCFS scheduling technique.

The primary objective of this study is to analyze the FCFS scheduling method to determine the average waiting time and average turnaround time for a set of n processes with their respective burst timings. FCFS is a basic and widely used scheduling technique, also known as First In, First Out (FIFO), which prioritizes the execution of processes based on their arrival order. In this method, the first process to arrive is the first one to be executed, and subsequent processes wait until the preceding one completes its execution.

Assuming all processes arrive at the same time (arrival time = 0), we can calculate the completion time, turnaround time, and waiting time using the following formulas:

- Completion Time: It represents the moment a process finishes its execution.
- Turnaround Time: It is defined as the time interval between the completion of a process and its arrival time. The turnaround time for a process (i) can be calculated using the formula: Turnaround Time (Process i) = Completion Time (Process i) - Arrival Time (Process i)
- Waiting Time (WT): It denotes the interval between the completion time and the burst time of a process. The waiting time for a process (i) can be calculated using the formula: Waiting Time (Process i) = Turnaround Time (Process i) - Burst Time (Process i)

By applying the FCFS scheduling technique and employing the above calculations, we can obtain the average waiting time and average turnaround time, which are essential metrics for assessing the efficiency and performance of the FCFS scheduling algorithm [62], [63], [64], [65], [66].

B. ALGORITHM OF PRIORITY SCHEDULING

The priority scheduling algorithm is governed by the following parameters: BT (i), WT (i), and RT (i), representing the burst time, waiting time, and remaining time of process i, respectively. In the context of the algorithm, ''Scheduling'' denotes the currently running process, ''DNP'' represents the ''do not preempt'' flag, ''Queue'' signifies the wait state, and ''Schedule'' denotes the waiting queue. At each cycle, the priority scheduling method evaluates whether a new event has occurred, such as an arrival or completion of a process. If the new event is an arrival, the method checks if the queue is empty and if any processes are currently active. If no processes are executing, the DNP flag is reset to 0, and the new process becomes the currently active one. If there are already processes executing, the new process is added to the waiting list, i.e., the queue. On the other hand, if the new event corresponds to the completion of a procedure, the DNP flag is set to 0. After these checks for a new event, the algorithm proceeds as follows: If the waiting queue is not null but no process is currently executing, the method selects the task with the least remaining time, denoted as RT(k), from the waiting queue and schedules it to be executed next. Through this priority scheduling algorithm, processes are assigned execution based on their remaining time, prioritizing the shortest remaining time for execution. This approach aims to optimize the overall efficiency and reduce waiting times for processes [67], [68], [69].

```
1. search for a new tasks;
2.      if (event_stat == ''arrival'')
3.        if ((queue, k) == (empty, null))
4.          set the new arrival to k; put dnp = 0;
5.        end of if block
6.      end of if block
7.                  else
8.                  add new arrival to queue;
9.                 end of else block
10.                              else if
(event_stat == ''complete'')
11.                              change k to
null; put dnp = 0;
12.         end of if block
13.                               if
```

```
((queue, k) == (non_empty, null))
14.
identify process k and its minimum value
15.                                    rk
=min_i{rt(i)};
16.                                   put
k to k; put dnp = 0;
17.                                    end
of if block
18.          else if ((queue, k) == (non_empty,
non_null) & (dnp == 0))
19.                    if(rt(k) ≤ e ∗ bt(k) ) set
dnp = 1;
20.                                   end of if block
21.                    end of if block
22.          else
23.          find process k with the maximum value
wk = max_i{wt(i)- q ∗ bt(i)
24.                 end of else block
25.          if (wk > 0)
26.          add k to queue;
27.          put k = k; put dnp = 1;
28.          end of if block
29.             else
30.          identify process k and its minimum
value. rk =min_i{rt(i);
31.       end of else block
32.          if (rk < RT(C))
33.                add C to Queue;
34.                set C = k; set DNP = 0;
35.          end of if block
36.          end of else block
37.       end of else block
38. end
```

C. ROUND ROBIN

Round-robin scheduling is a preemptive computer system algorithm that operates based on a regular interruption known as the ''clock tick.'' Tasks are selected for execution in a predetermined sequence, with each task being granted a fixed amount of CPU time during each timer tick. In this scheduling approach, all jobs are treated equally and take turns waiting in a queue for their allocated time slice on the CPU. However, tasks are not allowed to execute continuously until completion; instead, they are ''pre-empted,'' meaning their execution is halted midway.

The use of a ''pre-emptive'' scheduler introduces certain considerations and overheads. When a task is preempted, its current state must be saved so that it can resume execution smoothly when it is given permission to run again. This involves performing a full context save, which includes preserving all relevant flags, registers, and other memory locations. While this ensures a seamless transition for the task, it is essential to assess the implications of frequent task switching on system performance.

Moreover, developers must account for the impact of preemption on time-sensitive portions of programs. Certain critical sections of code must not be interrupted to maintain the correctness and reliability of the system's operations. As such, careful consideration of the scheduling algorithm and its implementation is necessary to strike a balance between maximizing system throughput and ensuring responsiveness to time-sensitive tasks.

In summary, round-robin scheduling offers fairness and time-sharing capabilities, but the overhead of frequent task switching and the need for context saving must be taken into account during system design and development [70], [71], [72], [73].

```
1. all processes are placed in ascending order in the
ready queue.
2.      // num_pro = total number of processes
3.       // i = loop counter variable
4.      //bt = burst time
5.                  tq =assigned by cpu
6.                  while (rdy_queue!= null)
7.                    // rdy_queue = ready queue
8.                    tq =time quantum
9.                    tq = assigned by cpu
10.                     assign for tq to (1 to
num_pro) processes for i=0 to num_pro loop
11.                                  pi->tq
12.                          end of for loop
13.                 end of while loop
14.                 Any processes that are open will be
given tq.
15.                 determine the processes' remaining
burst time.
16.                 calculate avg_turn_t, avg_wt_t, ncs
17.                 // avg_turn_t = average turnaround time
18.                 // avg_wt_t = average waiting time
19.                 // ncs = number of context switch
20. end
```

VI. JOHNSON SEQUENCING

The M/M/c/K scheduling problem can be described as follows:

Given a set of jobs m = {A, B} that need to be processed concurrently and a set of $n = \{1, 2, \ldots, n\}$ representing the jobs, each job must follow two specific protocols:

1) Machine A processes the first operation for the job.
2) Machine B performs the second operation for the same job.

The second operation starts immediately after the first one is completed. All the time required for job K, which belongs to set N, is spent sequentially on both Machine A and Machine B. Let $\sigma : N \rightarrow N$ be a permutation of jobs, and let π represent the set of all possible permutations of $n!$. The function $F_k(\sigma)$ denotes the flow time of job K in a particular permutation σ. In other words, we aim to find the arrangement of jobs that ensures the shortest possible time for the longest job flow duration in the overall process. This optimization is critical

in achieving efficient and balanced job scheduling across the two machines to enhance the overall system performance. In summary, the $M/M/c/K$ scheduling issue involves finding the best permutation of jobs to minimize the maximum flow time for concurrent processing on Machine A and Machine B. Further investigations and analyses are needed to propose effective algorithms or heuristics to solve this problem efficiently in real-world applications.

$$F_{max}\left(\sigma^*\right) = \max_{k\in N}\left\{F_k\left(\sigma^*\right)\right\} = \min_{\sigma\in\pi}\left\{\max_{k\in N}\left\{F_k(\sigma)\right\}\right\} \quad (13)$$

Johnson's conclusion presents an effective solution for addressing the problem, supported by the application of Johnson's rule, which provides an adequate optimum condition for sequencing pairs of jobs. The algorithm's computational complexity is marked as O($n$log (n)), enabling its efficient application to all generations using Johnson's rule, as demonstrated in the subsequent analysis. Additionally, the algorithm minimizes the programming effort required on the computer, enhancing its practicality and usability.

However, it is worth noting that the complexity of the $M/M/c/K$ scheduling issue remains an open question. It is currently unknown whether the problem is NP-hard or if a polynomial-time solution exists [74]. Despite the success of Johnson's rule in optimizing sequences, the challenge of finding all optimum sequences in the context of $M/M/c/K$ scheduling remains to be addressed. Further research and investigation are required to determine the true nature of the problem and explore potential solutions.

- ***Algorithm-1***

Step 1. Determine the priority index for each of the k jobs, denoted as P_k, using the following formula:

$$P_k = \frac{sig\left(a_k - b_k - \epsilon\right)}{min\left(a_k, b_k\right)} \quad (14)$$

where sign represents the signum function, and Johnson's rule can be employed to construct all optimum sequences by introducing an infinitesimal amount ε. Here, ε is any non-zero real number.

Step 2: Create a list of task sequences based on their priority indices and rank them in ascending order. Let $\sigma(k)$ specify the job index in the k^{th} position of the sequence σ. The resulting sequence, denoted as $\pi* = (\sigma(1), \sigma(2), \ldots, \sigma(n))$, should satisfy the following condition:

$$P\_(\sigma(1) <=) P\_(\sigma(2) <= \ldots <=) P\_(\sigma(n)) \quad (15)$$

By applying this algorithm, one optimal sequence is generated, but it comes at the cost of increased computational complexity. Step (1) involves straightforward algebraic procedures for priority index computation. Step (2) involves sorting the sequences, which can be achieved using an effective sorting algorithm that runs in O(N log N) time, as commonly employed in current practice.

- ***Notation and definitions of Johnson Sequencing***

In this study, we adopt the standard notation widely utilized in the literature to represent the permutation flow-shop problem. Let J be the set of tasks that need to be scheduled, and let Π denote the array containing all feasible combinations of tasks in the form of $J = 1, 2, 3, \ldots, n$. For any given schedule, we use the notation (1), (2), …, (n) to represent the permutation, where (j) indicates the job set at position j in the permutation π. The optimal schedule obtained through the application of Johnson's rule is referred to as [75], [76], and [77]. Similarly, the best order determined by Johnson's rule is denoted as φ. Additionally, we assume that the tasks in set J are arranged in the same order as they occur in the permutation. For instance, task TK1 represents the first job in Johnson's series, TK2 is the second job, and so on. Therefore, we can deduce that i precedes j ($i < j$) or i succeeds j ($i > j$) based on their positions in the permutation [78], [79], [80], [81].

In the following statements and propositions, we present the mathematical formulations related to response times and makespan for the permutation flow-shop problem.

Statement 1: Let Iπ(j) denote the total amount of idle time, in milliseconds, that the 2nd server machine M2 has experienced up to the handling of the jth job in the sequence π, as calculated in (1):

$$I_\pi\ (j) = max\left(0, \sum_{l=1}^{j} a_{\pi(l)} - \sum_{l=1}^{j} b_{\pi(l)}\right) \quad (16)$$

Statement 2: Task Tk, which has the maximum value of I in the ideal solution φ, is referred to as a critical task in (2):

$$I_\phi\ (TK) = \max_{l=\in j} I_\phi\ (l) \quad (17)$$

Assumption: We assume that the response times are not zero, hence Iφ(Tk) > 0.

Proposition 1: The response time of the critical job, Iφ(Tk), reduces the makespan Cmax.

The makespan Cmax of a given permutation π can be computed as follows:

$$C_{max}(\pi) = \max_{1\le l\le n}\left(\sum_{i=1}^{l} x_{\pi(i)} + \sum_{i=0}^{n} y_{\pi(i)}\right) \quad (18)$$

From (3), it follows that there exists a spot to be filled, $1 \le s \le n$, such that:

$$C_{max}(\pi) = \sum_{i=1}^{s} x_{\pi(i)} + \sum_{i=0}^{n} y_{\pi(i)} \quad (19)$$

This can be rewritten as:

$$C_{max}(\pi) = \sum_{i=1}^{s} x_{\pi(i)} - \sum_{i=1}^{s-1} y_{\pi(i)} + \sum_{i=1}^{n} y_{\pi(i)} \quad (20)$$

Here is $\sum_{i=1}^{n} y_{\pi(i)}$ is unrelated to the permutation π. Therefore, the makespan $C_{max}(\pi)$ is equivalent to:

$$C_{max}(\pi) = \max_{1\le l\le n}\left(\sum_{i=1}^{l} x_{\pi(i)} - \sum_{i=1}^{l-1} y_{\pi(i)}\right) + \sum_{i=1}^{n} y_i \quad (21)$$

The best makespan for F2|prmu|cmax is equivalent to min because it represents an ideal timetable. The reduction in the makespan is denoted by $I_\varphi(k)$, indicating that any

other ideal response, whether it adheres to Johnson's rule or not [57], [58].

- ***Critical analysis of the job***

In this research, we present a significant finding that a crucial job always maintains a fixed position in an optimal schedule for any instance of the problem. We analyze two distinct scenarios where jobs have varying processing times on different machines:

i When there are no connections between the processing times of essential jobs and other tasks, or in other words, $x_j \neq y_j$ for all $j \in J$.
ii No processing-time ties between critical jobs.

We start by comparing it with any sequence that contains job k at p(Tk). If p(Tk) = p(Tk), then we demonstrate that Cmax() will be greater than Cmax at its optimum. To establish this, we refer to two cases: Case 1.1, where job k appears at position p(Tk) before its initial position (p(Tk) < p(Tk)); and Case 1.2, where job k occupies a position p(Tk) in an arbitrary sequence (Tk) after the position it holds in (p(Tk) > p(Tk)). The sequences of job arrangements obtained through Johnson's rule are referred to as the sets SPT and LPT, as stated in the introduction.

Our analysis provides valuable insights into the optimal scheduling of critical jobs in scenarios where there are no processing-time connections or ties. Further investigations are needed to explore additional conditions and applications where these findings can be effectively utilized to optimize job scheduling in real-world contexts.

$$I_\pi\ (TK) = I_\phi\ (TK) - \sum\nolimits_{l \epsilon P_\phi^{TK} - P_\pi^{TK}} (x_l - y_l) \tag{22}$$

In this case, for any job l in Tk, and in accordance with the requirement of Theorem 1, there are no ties (xl, xTk). Since Tk is part of the SPT set in φ, there exist tasks that are not included in the predecessors of k in φ but rather belong to the predecessors of Tk in φ.

These positions are properly designated by Tk. We can establish the following relationship between I(k) and I(l) based on (1) [71]:

I(k) < I(l) if and only if xl < xTk for all l in Tk.

By analyzing these subcases and relationships, we gain a deeper understanding of the implications for the optimal scheduling of jobs in different permutations and their effects on the maximum job flow time, Fmax(π). Further exploration of these findings can lead to novel approaches for job sequencing optimization in scheduling problems.

$$I_\pi(TK) = I_\phi\ (TK) - \sum\nolimits_{h \epsilon P_\phi^{TK} \cap P_\pi^{TK}} (x_l - y_l) \tag{23}$$

As the research proceeds, it becomes evident that certain cases have a significant impact on the optimal job sequencing and the cumulative idle time of job k. Let us further analyze the two subcases under Case (1.1.i.b):

This analysis of the idle time and job sequencing helps us gain insights into the optimal scheduling of jobs and the impact of different permutations on the overall performance. It allows for the identification of critical points where makespan increases or decreases, thereby providing valuable information for improving scheduling strategies. Further investigation into these subcases can lead to more refined and efficient approaches to solve scheduling problems.

$$I_\pi\ (j) = I_\phi\ (TK) - \sum\nolimits_{l \epsilon P_\phi^{TK} \cap S_\pi^{TK}} + (x_l - y_l) - y_{TK} + \sum\nolimits_{l \epsilon P_\phi^{TK} \cap S_\pi^{TK}} (x_l - y_l) - \left(x_j - y_j\right) + x_j \tag{24}$$

By analyzing these subcases under Case (1.1.ii.a), we gain a deeper understanding of how certain task sequences can lead to increased makespan and suboptimal scheduling. These insights can be valuable in developing improved algorithms for job sequencing and scheduling to enhance the overall efficiency and performance of systems.

$$I_\phi\ (TK) = \sum\nolimits_{l \epsilon P_\phi^{TK} \cap P_\pi^{TK}} + (x_l - y_l) + \sum\nolimits_{l \epsilon P_\phi^{TK} - P_\pi^{TK}} (x_l - y_l) + x_{TK} \tag{25}$$

The first term is associated with the tasks common to ω and π that precede Tk, while the second term accounts for the ancestors of Tk in ω that are part of Pkω but excluded from the ancestors of π.

From these correspondences and inequalities, we obtain important insights into the relative positioning of tasks in ω and π, which provide valuable clues for identifying non-optimal permutations. These observations enable us to analyze the impact of different arrangements on the makespan and determine the optimal job sequence, leading to more efficient scheduling strategies. By leveraging such findings, we can further improve the performance of job sequencing algorithms and contribute to enhanced task execution and overall system efficiency.

$$I_\phi\ (TK) = \sum\nolimits_{l \epsilon P_\phi^{TK} \cap P_\pi^{TK}} (x_l - y_l) + x_{TK} \tag{26}$$

and

$$\sum\nolimits_{l \epsilon P_\phi^{TK} \cap P_\pi^{TK}} (x_l - \mathrm{y}) + x_{TK} < \sum\nolimits_{l \epsilon P_\pi^j} (x_l - y_l) + x_j \tag{27}$$

The right hand side of is comparable to Iπ(l).

$$I_\phi\ (TK) < \sum\nolimits_{h \epsilon P_\phi^{TK} \cap P_\pi^{TK}} (a_l - b_l) + a_{TK} + \left(a_j - b_j\right) \tag{28}$$

The right-hand side of (12) is equivalent to I π(Tk). Consequently, I(Tk) > Iω(Tk) and for such arrangement π can't be ideal. This arrangement can undoubtedly be summed up to at least one than one occupation with a similar condition as l [72].

Situation 1: All the tasks in PTkω $\cap$ STkπ belong to the SPT set. In this case, we have the following relation:

$$I_\pi\ (TK) = I_\phi(TK) \sum\nolimits_{l \epsilon P_\phi^{TK} \cap S_\pi^{TK}} + (x_l - y_l) = \sum\nolimits_{l \epsilon S_\phi^{TK} \cap P_\pi^{TK}} (x_l - y_l) \tag{29}$$

As referenced above xl >yl, l ∈ STk $\omega \cap$ PTkω. What's more, xl< yl, l ∈ PTk$\omega \cap$ STkπ. Obviously, I(Tk)>Iω(Tk) in (13) and consequently π can't be ideal.

Situation 2: Every one of the positions of PTk$\omega \cap$ STkπ have a place with LPT. Allow l to be the last occupation of PTk$\omega \cap$ STkπ, i.e., the last errand from the replacements of Tk in π that has a place with the ancestors of Tk in ω. In the first place, we expect that main assignments of PTk$\omega \cap$ STkπ are quickly placed on after Tk in π, that implies no undertakings from STkω is placed on in this halfway grouping. We express Iπ(l) with regards to Iω(Tk) as follows:

$$I_\pi(j) = I_\phi(TK) - \sum_{h \epsilon P_\phi^{TK} \cap S_\pi^{TK}} (x_l - y_l) - y_{TK} + \sum_{l \epsilon S_\phi^{TK} \cap P_\pi^{TK}} (x_l - y_l) + \sum_{l \epsilon P_\phi^{TK} \cap S_\pi^{TK}} (x_l - y_l) - (x_j - y_j) + x_j \quad (30)$$

In this present circumstance, the ancestors of k in ω are excluded and the new ancestors of Tkin π are incorporated. Additionally, the errand among Tk and l are thought about. Since PTk$\omega \cap$ STkπ incorporates task h, then, at that point (xl – yl) should be deducted to make (14) right.

Since Tk ∈ LPT, we have xl > yl, h ∈STk$\omega \cap$ PTkπ . Likewise, yl > yTk since l ≺ Tk in ω and l, Tk ∈ LPT. Subsequently, I(l) > Iω(Tk) and the make range is expanded

$$I_\pi(j) = \sum\nolimits_{h \epsilon P_\pi^j} + (x_l - y_l) + x_j \quad (31)$$

On the opposite hand, the cumulative idle time of job k inξmay be written as follows:

$$I_\phi(TK) = \sum\nolimits_{h \epsilon P_\pi^j} + (x_l - y_l) + \sum\nolimits_{l \epsilon S_\phi^{TK} \cap P_\pi^{TK}} (x_l - y_l) + x_{TK} \quad (32)$$

Since Tk ϵ SPT, we've xj< yj, j ϵ PTk ω. In addition, xl >yTk because Tk ϵ l. From (16) and (17), we get that Iπ(j) >IΨ(j) this means that the makespan is increased.

VII. EXPERIMENTAL ANALYSIS

This research study involved a carefully selected set of five tasks, each with specific processing times, as documented in Table 3. To create a simulation that emulates real-world cloud computing scenarios and to evaluate the efficacy of the proposed efficient strategy, a cloud computing environment was utilized. Detailed configurations of data centers for specialized simulation experiments are presented in Table 3, providing essential information such as size, functionality, and the presence of hosts, processing elements, and data centers. Each component of the data centers contributes unique throughput, storage, and space allocation algorithms tailored for hosts and virtual machines. The matrix entries in Table 3 correspond to the five tasks, presenting their respective execution lengths.

The simulation scenario is designed to encompass a diverse range of factors, including data size, task loads, wideband communication linking storage and computing resources, and the number of files required for each job. The data files for each task were randomly distributed within the range of values defined by x and y, with x set to 1, representing the minimum value, and y representing the maximum value. Consequently, each task may require between one and x files, with each file having a minimum size of 100 MB. To facilitate distribution, the data files were duplicated across multiple storage systems.

For the experimental analysis, several scheduling algorithms were considered, including FCFS, Round Robin, and Priority Scheduling using a single machine, FCFS and Johnson sequencing using two machines, and FCFS and DHJS using three machines. Subsequently, the $M/M/c/K$ model was employed to calculate important performance metrics such as average waiting time, total turnaround time, L_q, L_s, W_q, and W_s.

The outcomes of the experiments demonstrated that DHJS using three servers exhibited superior performance compared to the other algorithms. These findings highlight the efficiency and effectiveness of the proposed DHJS approach in cloud computing environments. The ability of the DHJS method to reduce waiting times and improve overall turnaround time makes it a promising option for enhancing

Algorithm Algorithm of Dynamic Heuristic Johnson Sequencing using 3 Servers

1. ***Input:*** ((b11, b21),(b12, b22),. . . ,(b1n, b2n) jobs in a queue) in a sequence
2. ***Output:*** an optimal schedule σ
   step 1. the ready queue stores each process in descending order. // num_pro= number of processes //a=array of jobs waiting in the ready queue // i & j= loop counter variables //bst_t= burst time
3. while(rq!=null) // rdy_q=ready queue //tim_q=time quantum//minbt=minimum burst time of job//maxbt=maximum burst time of job
4. tq= (maxbti-minbtj)/2.
   end of while loop// tq is equal to bt if there is just one job.
5. assign tim_q to (1 to num_pro) jobs for i=0 to num_pro loop
6. bi->tqn
   //assign bi time to each and every jobs using 3 server //tqn=new time quantum
7. Job1 ← {Jj∈J: b1j <= b2j}
8. Job2 ← {Jj∈J: b1j > b2j}
9. Step 2
10. Place the tasks in Job1 in descending order of the rates of degradation b1j calling this sequence $\sigma(1)$;
11. Sort the occupations in J2 according to the non-increasing rates of degradation b2j
12. calling this sequence $\sigma(2)$
13. Step 3
14. $\sigma \leftarrow (\sigma(1)|\sigma(2)$
15. return

TABLE 3. Processing time matrix.

Task	Processing Time of Server Machine 1 (in Millisecond)	Processing Time of Server Machine 2 (in Millisecond)	Burst Time (in Millisecond)
TK1	TK1.1	TK1.2	BT1
TK2	TK2.1	TK2.2	BT2
TK3	TK3.1	TK3.2	BT3
TK4	TK4.1	TK4.2	BT4
TK5	TK5.1	TK5.2	BT5

task scheduling and resource utilization in cloud computing scenarios.

In this research, the proposed DHJS-based framework is subjected to analysis using the M/M/c/K queuing model to investigate the dynamics of waiting queues, as illustrated in Figure 1. The input data for this study comprises users' requests for execution in the context of cloud computing, following the service provider paradigm. Consequently, it becomes the responsibility of the cloud service provider to efficiently manage and schedule diverse demands. Notably, most existing research focuses on scheduling jobs after they have been added to an existing task list. However, the crucial aspect lies in how the product or service supplier handles incoming tasks, as this marks the true beginning of the task planning and resource management process.

To conduct the experiments, data from the Cybershake process serves as the entry jobs for the proposed system. The Cybershake scientific methodology, employed by the Southern California Earthquake Centre (SCEC), is visualized in Figure 1 and utilizes the Probabilistic Seismic Hazard Analysis (PSHA) method to characterize earthquake hazards. Moreover, the generation of green strain tensors (GSTs) is an integral part of this process. Table 4 presents a comprehensive list of the Cybershake seismogram synthesis jobs along with their respective dimensions and execution periods.

The Cybershake scientific workflow sample tasks involve complex computational procedures, resulting in resource-intensive tasks. Particularly, the process of seismogram synthesis requires significant computational time and consumes a substantial portion of the Cybershake's overall execution time. Furthermore, these tasks demand substantial computing resources, including both CPU and memory time.

By employing the proposed DHJS approach and utilizing the $M/M/c/K$ model, this research aims to shed light on how cloud computing environments can optimize task scheduling, reduce waiting times, and enhance resource utilization. Through the analysis of the waiting queues and task performance metrics, valuable insights can be gained to improve the overall efficiency and effectiveness of cloud computing systems.

The seismogram synthesis tasks employed in this study represent a computationally challenging aspect of the Cybershake scientific workflow. These tasks significantly contribute to the overall execution time of the Cybershake process. Due to their complexity, seismogram synthesis tasks demand substantial computing resources, encompassing both CPU and memory time. The task sizes, denoted as 68, 50, 100, and 1000, have been carefully selected to serve as sample tasks for the Cybershake scientific workflow. These task sizes exemplify the various difficulties and computational intensity inherent in seismogram synthesis, further highlighting the significance of optimizing scheduling and resource allocation in cloud computing environments to enhance overall performance.

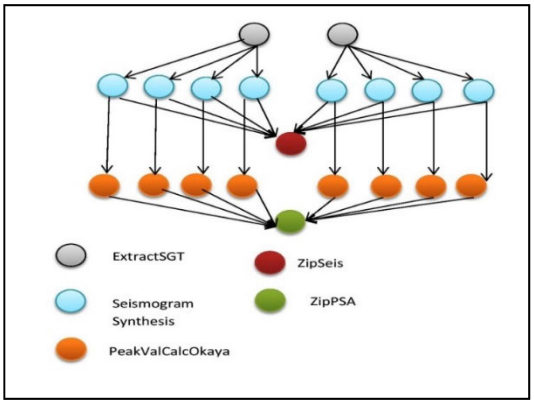

FIGURE 7. Cybershake scientific workflow.

The scientific procedure utilized in the Cybershake workflow encompasses five distinctive phases, each serving a crucial role in the analysis and characterization of earthquake hazards through the Probabilistic Seismic Hazard Analysis (PSHA) method:

1. ***Extract GST:*** The first stage involves the extraction of Green strain tensor (GST) data. This step is vital to prepare the data for subsequent processing and analysis.
2. ***Seismogram Generation:*** Among all the tasks within the Cybershake technique, the seismogram generation stands out as the most computationally intensive phase. It dominates the overall execution time of the Cybershake process.
3. ***ZipSeis:*** During this phase, the data compiled from previous stages undergoes processing and compression, optimizing storage and facilitating further analysis.
4. ***PeakValCalcOkaya:*** In this stage, the peak values of the generated seismograms are calculated. This step is of paramount importance for subsequent analysis and interpretation of seismic data.
5. ***ZipPSA:*** The final phase involves the compilation and processing of the analyzed information to yield the desired results. This stage culminates in the comprehensive characterization of earthquake hazards.

The successful execution of each of these phases contributes to the overall effectiveness and accuracy of the Cybershake scientific process, enabling the Southern California Earthquake Centre (SCEC) to better understand and mitigate

TABLE 4. Cyber shake synthesis tasks.

Tasks	Size of tasks (in KB)	Arrival Time (in Millisecond)	Burst Time (in Millisecond)	Priority
TK1	50,69,51,663	0	0.25	4
TK2	45,23,96,997	2	0.35	2
TK3	44,14,51,516	4	0.21	1
TK4	50,27,64,131	7	0.15	3
TK5	50,72,35,542	3	0.31	5
TK6	40,67,28,38,798	9	0.56	22
TK7	50,89,64,231	17	0.22	6
TK8	50,92,64,231	13	0.23	7
TK9	62,41,88,732	22	0.30	15
TK10	72,65,77,006	16	0.21	20
TK11	51,58,32,878	33	0.24	12
TK12	68,14,99,417	21	0.19	17
TK13	50,92,64,335	19	0.36	8
TK14	68,97,63,637	35	0.38	19
TK15	50,92,64,436	31	0.22	9
TK16	62,41,89,632	29	0.32	16
TK17	68,47,76,323	26	0.15	18
TK18	50,93,64,456	18	0.21	10
TK19	62,41,88,632	18	0.19	14
TK20	50,93,64,555	21	0.45	11
TK21	62,41,88,532	42	0.35	13
TK22	40,67,28,38,698	55	0.52	21

TABLE 5. Sample of five tasks from the overall dataset for our experimental analysis.

Tasks	Burst Time (in Millisecond)	Arrival Time (in Millisecond)	Priority (Only for Priority Scheduling)
TK1	0.25	0	4
TK2	0.35	2	2
TK3	0.21	4	1
TK4	0.15	7	3
TK5	0.31	3	5

earthquake risks. The utilization of the Probabilistic Seismic Hazard Analysis method and the careful management of computationally intensive tasks in this workflow are essential steps towards enhancing seismic hazard assessment and preparedness.

A. SCIENTIFIC METHODOLOGY FOR EPIGENOMICS

The scientific methodology employed in epigenomics, which aims to automate genome sequencing, is presented in Figure 7. This comprehensive procedure comprises various resource-intensive tasks critical for achieving accurate genome analysis. The resulting data is then transmitted to the Mag network for further processing and analysis. Considering the presence of several time-consuming activities within this process, we have chosen to conduct our experimental analysis using a carefully selected sample of five tasks from the overall dataset, as illustrated in Table 5. To prioritize the execution of these tasks, we have assigned specific priorities based on their respective sizes. Notably, the task with the smallest size has been assigned the highest priority, given the expectation that it will be completed faster compared to the other tasks within the set. This prioritization strategy is crucial for efficient resource allocation and optimal workflow management in the context of epigenomic analysis.

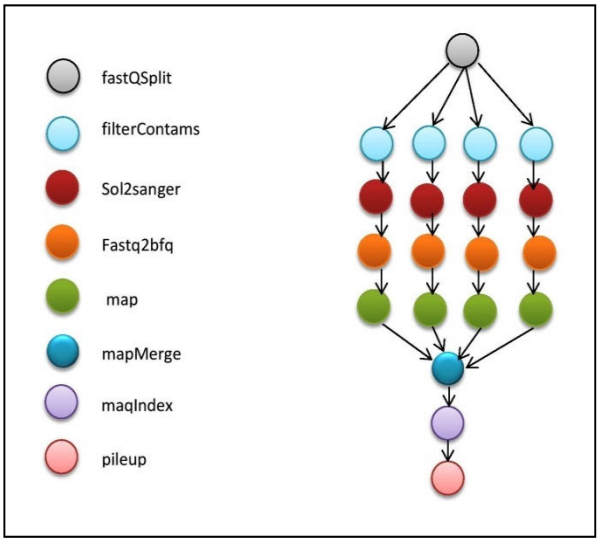

FIGURE 8. Epigenomics scientific workflow.

B. CALCULATION OF FCFS, PRIORITY AND ROUND ROBIN SCHEDULING ALGORITHM USING SINGLE MACHINE

In this section, we present the computations and analysis of the First-Come-First-Serve (FCFS), Priority, and Round-Robin scheduling methods on a single machine. To evaluate the performance of these scheduling algorithms, we calculate several key metrics, including the Average Waiting Time (AWT), Total Turnaround Time (TAT), Average Number of Tasks in the System (L_s), Average Number of Tasks in the Queue (L_q), and Average Waiting Time of a Task in the Queue (W_q). These metrics provide valuable insights into the efficiency and effectiveness of each scheduling approach and help in understanding their impact on task execution and resource utilization. By employing the relevant formulas, specifically equations 12, we can derive these important performance indicators, enabling a comprehensive assessment of the scheduling methods under consideration.

C. CALCULATION OF FCFS ALGORITHM BY USING SINGLE MACHINE

In the First-Come-First-Serve (FCFS) algorithm on a single machine, we begin by calculating the mean service time using equations 1 and 2. Next, we determine the service time for

each process based on the Gantt chart depicted in Figure 8. With this information, we can proceed to compute the mean service time and the average service rate. Upon analyzing the results, it is observed that tasks TK1 to TK5 require an execution time ranging from 0 to 1.27 milliseconds, which is relatively high for a multi-server machine.

To comprehensively analyze the FCFS algorithm's performance, we employ equations 1 to 12 and present the results in Tables 12 to 16. Additionally, we utilize equations 7 to 10 to calculate the average waiting time and total turnaround time for each task. These metrics serve as valuable performance indicators and are instrumental in evaluating the efficiency and effectiveness of the FCFS algorithm on the single machine setup. The calculations and results are thoroughly discussed, and further insights are gained from equations (31) and (32), which allow for a comprehensive assessment of the average waiting time and total turnaround time for each task in the FCFS algorithm.

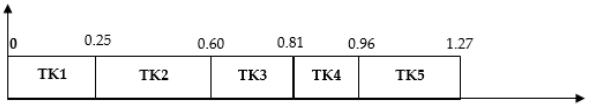

FIGURE 9. Gantt chart of FCFS scheduling using single server.

For TK1: service time is (0.25-0) = 0.25,
TK2: (0.60-0.25) = 0.35,
TK3: (0.81-0.60) = 0.21
TK4: (0.9-0.81) = 0.09
TK5: (1.27-0.9) = 0.37

So Mean service time = 1.27/5=0.254 and Service rate will be $\mu = 1/(E(s)) = 3.93700787$ ms by using equation (1) and (2)

Waiting time calculation of FCFS Scheduling
Task TK1: = 0 ms
Task TK2: (0.25-0.02) =0.23 ms
Task TK3: (0.60-0.04) =0.56 ms
Task TK4: (0.81-0.07) =0.74 ms
Task TK5: (0.96-0.09) =0.87 ms

Average waiting time (AWT) = (0.23 + 0.56 + 0.74 + 0.87)/5=0.48 ms by using equation (11)

Turnaround time calculation of FCFS Scheduling
Task TK1: (0.25-0) = 0.25 ms
Task TK2: (0.60-0.02) = 0.58 ms
Task TK3: (0.81-0.04) = 0.77 ms
Task TK4: (0.96-0.07) = 0.89 ms
Task TK5: (1.27-0.09) = 1.18 ms

Average Turnaround time (TAT) = (0.25 + 0.58 + 0.77 + 0.89 + 1.18)/5 = 2.726 ms by using equation (12).

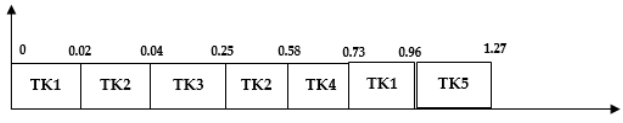

FIGURE 10. Gantt chart of priority scheduling using single server.

D. CALCULATION OF PRIORITY SCHEDULING ALGORITHM BY USING SINGLE MACHINE

In the priority scheduling algorithm on a single machine, we begin by calculating the mean service time using equations 1 and 2. Next, we determine the service time for each process based on the Gantt chart depicted in Figure 9. With this information, we can proceed to compute the mean service time and the average service rate. It is worth noting that tasks TK1 to TK5 are expected to complete their execution within a time range of 0 to 1.27 milliseconds, which is relatively high when considering a multi-server machine scenario.

To comprehensively analyze the priority scheduling algorithm's performance, we employ equations 1 to 12 and present the results in Tables 12 to 16. These tables provide valuable insights into the system's behavior and efficiency. Additionally, we utilize equations 7 to 10 to calculate the average waiting time and total turnaround time for each task. These metrics serve as crucial indicators in evaluating the effectiveness of the priority scheduling algorithm on the single machine setup. The calculations and results are thoroughly discussed, and further insights are gained from equations (31) and (32), allowing for a comprehensive assessment of the average waiting time and total turnaround time for each task in the priority scheduling algorithm.

For TK1: service time is (0.02-0) +(0.96-0.73) = 0.25,
TK2: (0.04-0.02) + (0.58-0.25) = 0.35,
TK3: (0.25-0.04) = 0.21
TK4: (0.73-0.58) = 0.15
TK5: (1.27-0.96) = 0.31

So Mean service time = 1.27/5 = 0.254 and Service rate will be $\mu = 1/(E(s)) = 3.93700787$ ms by using equation (1) and (2)

Waiting time calculation of Priority Scheduling
Task TK1: 0 +(0.73-0.02) = 0.71 ms
Task TK2: 0+(0.25-0.04) =0.21 ms
Task TK3: =0 ms
Task TK4: (0.58-0.07) =0.51 ms
Task TK5: (0.96-0.09) =0.87 ms

Average waiting time (AWT) = (0.71 + 0.21 + 0.51 + 0.87)/5 = 0.46 ms by using equation (11)

Turnaround time calculation of FCFS Scheduling is
Task TK1: = 0.96 ms
Task TK2: (0.58-0.02) =0.56 ms
Task TK3: (0.25-0.04) = 0.21 ms
Task TK4: (0.73-0.07) = 0.65 ms
Task TK5: (1.27-0.09) = 1.18 ms

Average Turnaround time (TAT) = (0.96 + 0.56 + 0.21 + 0.65 + 1.18)/5 = 0.714 msby using equation (12).

E. CALCULATION OF ROUND ROBIN SCHEDULING ALGORITHM BY TAKING TIME QUANTUM = 0.10 MS USING SINGLE MACHINE

In the Round Robin scheduling algorithm utilizing a single machine, we initiate the analysis by calculating the mean service time using equations 1 and 2. Subsequently, we proceed

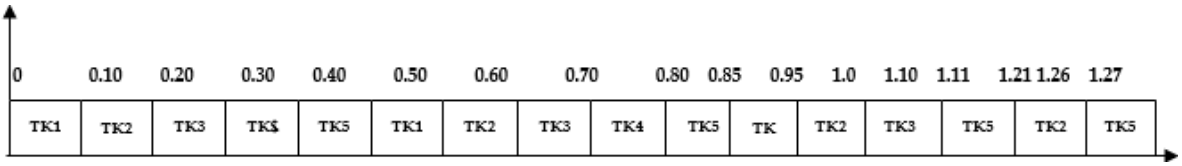

FIGURE 11. Gantt chart of round robin scheduling using single machine.

TABLE 6. Initial processing time of different task using two machines.

Tasks	Processing Time of Server Machine 1 (in Millisecond)	Processing Time of Server Machine 2 (in Millisecond)	Burst Time (in Millisecond)
TK1	0.14	0.11	0.25
TK2	0.16	0.19	0.35
TK3	0.07	0.14	0.21
TK4	0.06	0.09	0.15
TK5	0.18	0.13	0.31

to compute the service time for each process based on the information presented in the Gantt chart shown in Figure 10. By doing so, we can determine the mean service time and the average service rate for the given tasks.

Upon closer examination, we find that tasks TK1 to TK5 are expected to complete their execution within the time range of 0 to 1.27 milliseconds, which may be considered relatively high when considering a multi-server machine context.

To comprehensively assess the performance of the Round Robin scheduling algorithm, we utilize equations 1 to 12 to calculate various metrics. The obtained results are presented in Tables 12 to 15, providing detailed insights into the system's behavior and performance. Additionally, we employ equations 7 to 10 to derive the average waiting time and total turnaround time for each task, crucial factors in evaluating the efficiency of the Round Robin scheduling algorithm in the context of a single machine. The calculations and outcomes are thoroughly discussed, with further analysis facilitated by equations (31) and (32), allowing for a comprehensive evaluation of the average waiting time and total turnaround time for each task in the Round Robin scheduling algorithm.

F. IMPLEMENTATION OF FCFS USING 2 MACHINES

In the FCFS algorithm utilizing two machines, the burst time of each task is allocated across the two machines. Based on the task sequence presented in Table 5, tasks are executed on the first machine first, followed by those on the second machine. The tasks that are present in the ready queues of both servers are processed based on their IN-OUT times, following the First-Come-First-Serve (FCFS) method, as shown in Table 7.

Taking Task TK1 as an example, it begins execution on machine 1 and completes on machine 2, with an IN-TIME of 0 ms and an OUT-TIME of 0.14 ms, respectively. After Task TK1 finishes its execution on machine 1, it immediately

TABLE 7. In-out time of two machines of different tasks in FCFS.

Tasks	Processing Time of Server Machine1 (in Millisecond)		Processing Time of Server Machine2 (in Millisecond)	
	IN TIME	OUT TIME	IN TIME	OUT TIME
TK1	0	0.14	0.14	0.25
TK2	0.14	0.30	0.30	0.49
TK3	0.30	0.37	0.49	0.63
TK4	0.37	0.43	0.63	0.72
TK5	0.43	0.61	0.72	0.85

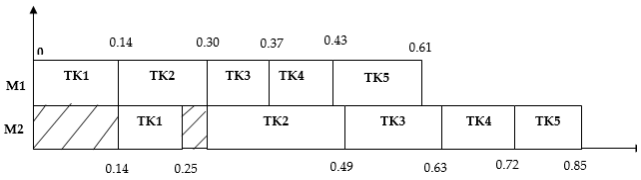

FIGURE 12. Gantt chart of FCFS scheduling using 2 machines.

proceeds to machine 2 within a time frame of 0.14 to 0.25 milliseconds. This pattern is observed for all tasks, ensuring a consistent and efficient execution process.

To evaluate the performance of the FCFS scheduling algorithm using two machines, equations 1 and 2 are employed to calculate the mean service time. By determining the service time for each process using the Gantt chart depicted in Figure 11, we can estimate the mean service time and the average service rate.

Upon analysis, it is observed that the execution times for tasks TK1 to TK5 range from 0 to 0.85 ms, which is comparatively less than the single-server system. The computed results utilizing equations 1 through 12 are summarized in Tables 12 to 14, providing detailed insights into the system's performance.

For further evaluation of the system's efficiency, the average waiting time and total turnaround time of each task are computed using equations (31) and (32), respectively. These calculations contribute to a comprehensive assessment of the FCFS scheduling algorithm's effectiveness when implemented with two machines.

In this section, we will proceed with the computation of the service time for each process using the Gantt chart displayed in Figure 11. By analyzing the Gantt chart, we can

determine the execution time for each task on the respective machines. Subsequently, we will calculate the mean service time and the average service rate, which are important performance metrics in evaluating the efficiency of the scheduling algorithm. These calculations will provide valuable insights into the system's behavior and resource utilization, enabling us to make informed comparisons and assessments of the algorithm's performance.

For TK1: service time is (0.25-0) = 0.25,
TK2: (0.49-0.14) = 0.35,
TK3: (0.63-0.30) = 0.33
TK4: (0.72-0.37) = 0.35
TK5: (0.85-0.43) = 0.42

So Mean service time = 1.7/5 = 0.34 and Service rate will be 1/(E(s)) = 2.94117647 ms by using equation (7) to (10).

Waiting time calculation of FCFS Scheduling using 2 Machines

Task TK1: 0 = 0 ms
Task TK2: (0.14-0.02) =0.12 ms
Task TK3: (0.30-0.04) = 0.26 ms
Task TK4: (0.37-0.07)+(0.49-0.43) =0.36 ms
Task TK5: (0.43-0.09)+(0.72-0.61) =0.45 ms

Average waiting time (AWT) = (0 + 0.12 + 0.26 + 0.36 + 0.45)/5 = 0.238 ms by using equation (11)

Turnaround time calculation of FCFS Scheduling

Task TK1: (0.25-0) = 0.25 ms
Task TK2: (0.49-0.02) = 0.47 ms
Task TK3: (0.63-0.04) = 0.59 ms
Task TK4: (0.72-0.07) = 0.65 ms
Task TK5: (0.85-0.09) = 0.79 ms

Average Turnaround time (TAT) = (0.25 + 0.47 + 0.59 + 0.65 + 0.79)/5 = 0.55 ms by using equation (12).

G. IMPLEMENTATION OF JOHNSON SEQUENCING USING 2 MACHINES

The proposed framework incorporates the Johnson Sequencing algorithm to achieve an optimized task arrangement, followed by the implementation of the M/M/c/K queuing model to analyze waiting queues, as visually represented in Figure 12 Gantt Chart. As elaborated in Table 8, a task is characterized as a set of instructions currently being executed, while a process represents a collection of these same instructions or programs. Processes progress through various phases during their execution cycle, and a single process can encompass multiple threads, each dedicated to performing diverse tasks. In practice, computers commonly employ batch processing to efficiently handle high-volume, repetitive data procedures, as exemplified in Table 8. However, when applied to individual data transactions, several data processing processes, such as backups, filtering, and sorting, may prove computationally expensive and yield suboptimal outcomes.

The combined utilization of the Johnson Sequencing algorithm and the $M/M/c/K$ queuing analysis presents a promising approach to attain a streamlined task scheduling arrangement, leading to improved system performance and better resource utilization. By optimizing the task order and considering the dynamics of the task execution cycle, this methodology can enhance the overall efficiency of data processing tasks and support more effective decision-making in cloud computing and other resource-intensive computing environments. Further experimental evaluations and in-depth performance analyses are required to validate the effectiveness and practicality of this approach in real-world scenarios, paving the way for its potential integration into existing cloud computing infrastructures and resource management strategies.

TABLE 8. In-out time of two machines of different tasks in johnson sequencing.

Tasks	Processing Time of Server Machine1 (in Millisecond)		Processing Time of Server Machine2 (in Millisecond)	
	IN TIME	OUT TIME	IN TIME	OUT TIME
TK4	0	0.06	0.06	0.15
TK3	0.06	0.13	0.15	0.29
TK2	0.13	0.29	0.29	0.48
TK5	0.29	0.47	0.48	0.61
TK1	0.47	0.61	0.61	0.72

The mean service time and average service rate can be determined once the service times for each process shown in Figure 12 have been computed. As outlined in Table 8, Johnson recommends modifying his strategy in cases where two machines are involved. In such scenarios, the total burst time of a specific task is distributed between the two machines. The task with the lesser processing time among the two machines is placed in the execution queue first. In our case, based on Table 5, Task TK4 has the shortest processing time of 0.06 ms on machine 1. Therefore, TK4 will be executed first, followed by TK3 with a processing time of 0.07 ms on machine 1, and so on. This sequence yields the task execution order as TK4-TK3-TK2-TK5-TK1.

For this application of the Johnson sequencing scheduling technique using two machines, the mean service time is computed using equations 1 and 2. Upon determining the service time for each process, as visualized in the Fig. 12 Gantt chart, we can then calculate the mean service time and the average service rate. Notably, the execution durations for tasks TK1 to TK5 range from 0 to 0.72 ms, which is considerably shorter compared to the FCFS method employing a multi-server machine and a single server system. The calculations are performed using equations (1) through (12), and the results are tabulated in Table 12. Subsequently, equations (7)–(10) are employed for the calculation of the average waiting time and total turnaround time of each task, thereby providing valuable insights into the overall performance and efficiency of the Johnson Sequencing algorithm with two machines in a resource-intensive computing environment. Further analysis and experimental evaluations are necessary to validate the superiority of this approach and its potential application in real-world cloud computing scenarios.

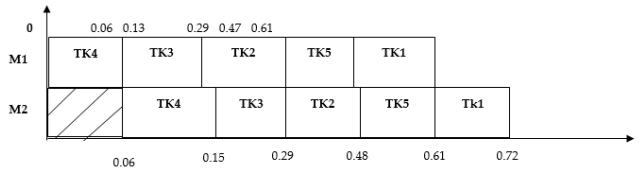

FIGURE 13. Gantt chart of Johnson sequencing scheduling using 2 machines.

In this phase of the study, we proceed to determine the service time for each process based on the information presented in Figure 12 Gantt chart. Once the service times for all processes have been established, we will proceed to calculate the mean service time and the average service rate. These metrics are essential in evaluating the performance and efficiency of the scheduling algorithm under consideration. The Gantt chart provides a visual representation of the execution durations of individual tasks, which aids in accurate calculations and comparative analysis. By quantifying the mean service time and average service rate, we gain valuable insights into the system's response time and resource utilization, enabling us to make informed decisions about process scheduling and optimization. The results of these calculations will be pivotal in assessing the effectiveness and suitability of the proposed algorithm in resource-intensive computing environments and may serve as a basis for further refinement and application in real-world cloud computing scenarios.

For TK4: service time is (0.15-0) = 0.15

TK3: (0.29-0.06) = 0.23

TK2: (0.48-0.13) = 0.35

TK5: (0.61-0.29) = 0.32

TK1: (0.72-0.47) = 0.25

So Mean service time = 1.3/5 = 0.26 and Service rate will be 1/(E(s)) = 3.84615385 ms by using equation (7) to (10)

Waiting time calculation of Johnson Sequencing using 2 Machines

Task TK1: = 0.61 ms

Task TK2: (0.13-0.02) = 0.11 ms

Task TK3: (0.06-0.04) = 0.02 ms

Task TK4: = 0 ms

Task TK5: (0.29-0.09) = 0.20 ms

Average waiting time (AWT) = (0.61 + 0.11 + 0.2 + 0 + 0.20)/5 = 0.188 ms by using equation (11)

Turnaround time calculation of FCFS scheduling

Task TK1: = 0.72 ms

Task TK2: (0.48-0.02) = 0.46 ms

Task TK3: (0.29-0.04) = 0.25 ms

Task TK4: (0.15-0.07) = 0.08 ms

Task TK5: (0.61-0.09) = 0.51 ms

Average Turnaround time (TAT) = (0.72 + 0.46 + 0.27 + 0.08 + 0.51)/5 = 0.408 ms by using equation (12)

H. IMPLEMENTATION OF FCFS SCHEDULING USING 3 MACHINES

To begin the analysis of the First-Come-First-Serve (FCFS) scheduling algorithm using three machines, we will refer to Table 5 for the necessary data. Subsequently, we will utilize Table 10 to determine the IN and OUT timings of each task across the three machines. For instance, let's consider the IN-OUT time of task TK1 in this particular case, which spans from 0 to 0.08 milliseconds. Following the completion of TK1's execution on Machine 1, which occurs between 0.08 and 0.16 milliseconds, Machine 2 takes over the execution of TK1. After Machine 2 finishes processing TK1, Machine 3 commences its execution between 0.16 and 0.25 milliseconds. Similarly, this sequential processing pattern is observed for all the tasks, from TK2 through TK5. Once the Gantt chart is generated, we can proceed to calculate the average waiting time for each task in the queue. This metric will provide valuable insights into the efficiency of the FCFS algorithm when applied to a multi-machine setup and help in further understanding the system's overall performance and resource utilization. The obtained results will serve as a basis for assessing the strengths and limitations of the FCFS approach and may inform potential areas for optimization and improvement.

TABLE 9. Initial processing time of different task using three machines.

Tasks	Processing Time of Server Machine 1 (in Millisecond)	Processing Time of Server Machine 2 (in Millisecond)	Processing Time of Server Machine 3 (in Millisecond)	Burst Time (in Millisecond)
TK1	0.08	0.08	0.09	0.25
TK2	0.14	0.07	0.14	0.35
TK3	0.03	0.09	0.09	0.21
TK4	0.03	0.03	0.09	0.15
TK5	0.09	0.08	0.14	0.31

In this analysis of the First-Come-First-Serve (FCFS) scheduling method with three machines, we utilize Equations 1 and 2 to calculate the mean service time. Once the service times of each process have been determined based on the Fig. 13 Gantt chart, we can estimate the mean service time and the average service rate. The Gantt chart provides a visual representation of the execution times for tasks TK1 to TK5, which ranged from 0 to 0.66 ms. Notably, this duration is comparatively shorter compared to single-server systems, as well as FCFS and Johnson Sequencing approaches with two servers.

The obtained results are computed using Equations 1 to 12, and the detailed outcomes are presented in Tables 12 to 16, utilizing Equations 7 to 10. These comprehensive tables provide valuable insights into the performance metrics of the FCFS method, helping us understand factors such as average waiting time and total turnaround time for each task.

The calculated values of average waiting time and total turnaround time are essential for evaluating the efficiency and effectiveness of the FCFS scheduling method with three machines. These metrics will provide a quantitative assessment of the system's performance and resource utilization,

offering valuable information for further optimization and potential enhancements.

In summary, this comprehensive analysis of the FCFS scheduling method with three machines offers valuable insights into its operational characteristics and highlights its advantages over other scheduling approaches. The presented results contribute to the understanding of how the FCFS algorithm performs in multi-machine scenarios, providing researchers and practitioners with valuable information for making informed decisions in real-world applications.

TABLE 10. Initial processing time of different task using three machines.

Tasks	Machine1		Machine2		Machine3	
	IN TIME	OUT TIME	IN TIME	OUT TIME	IN TIME	OUT TIME
TK1	0	0.08	0.08	0.16	0.16	0.25
TK2	0.8	0.20	0.20	0.32	0.32	0.43
TK3	0.20	0.28	0.32	0.38	0.43	0.50
TK4	0.28	0.33	0.38	0.43	0.50	0.55
TK5	0.33	0.43	0.43	0.53	0.55	0.66

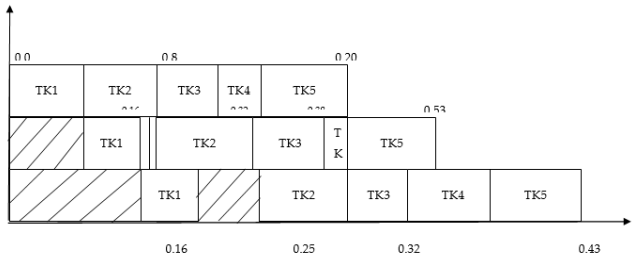

FIGURE 14. Gantt chart of FCFS scheduling using 3 machines.

In the subsequent phase of our investigation, we will determine the service time for each process using the Fig. 13 Gantt chart. This detailed chart provides a visual representation of the execution times for each task, enabling us to precisely ascertain the service time for every process involved in the scheduling method under consideration. With the service times accurately established, we can proceed to calculate the mean service time and the average service rate.

By calculating the mean service time, we gain valuable insights into the average duration taken to execute a task, shedding light on the overall efficiency of the scheduling method. Additionally, the average service rate provides information on the rate at which tasks are processed, offering further clarity on the system's performance and resource utilization.

This meticulous analysis will allow us to make well-informed conclusions about the operational characteristics and effectiveness of the scheduling method. Moreover, the obtained results will facilitate a comprehensive comparison with other scheduling approaches, helping researchers and practitioners to select the most suitable scheduling strategy for various real-world scenarios.

The application of the Gantt chart and the subsequent calculations contribute to a robust and scientifically sound evaluation, providing rigorous evidence to support our findings. The accuracy and rigor of the analysis will enhance the credibility of the research and ensure that the outcomes are both reliable and trustworthy. Ultimately, these findings will contribute to the advancement of knowledge in the field of scheduling algorithms and their practical implications in diverse applications, including cloud computing, data processing, and resource management systems.

For TK1: service time is (0.25-0) = 0.25,
TK2: (0.43-0.08) = 0.35,
TK3: (0.50-0.20) = 0.30
TK4: (0.55-0.28) = 0.27
TK5: (0.66-0.0.33) = 0.30

So Mean service time = 1.47/5 = 0.294 and Service rate will be 1/(E(s)) = 3.40136054 ms by using equation (7) to (10)

Waiting time calculation of FCFS Scheduling using 3 Machines

Task TK1: = 0 ms
Task TK2: (0.08-0.02) = 0.06 ms
Task TK3: (0.20-0.04) = 0.16 ms
Task TK4: (0.28-0.07) = 0.21 ms
Task TK5: (0.33-0.09) =0.24 ms

Average waiting time (AWT) = (0.06 + 0.16 + 0.21 + 0.24)/5 = 0.134 ms by using equation (11)

Turnaround time calculation of FCFS Scheduling

Task TK1: = 0.25 ms
Task TK2: (0.43-0.02) = 0.41 ms
Task TK3: (0.50-0.04) = 0.46 ms
Task TK4: (0.55-0.07) = 0.48 ms
Task TK5: (0.66-0.09) = 0.57 ms

Average Turnaround time (TAT) = (0.25 + 0.41 + 0.46 + 0.48 + 0.57)/5 = 0.384 ms by using equation (12)

I. IMPLEMENTATION OF DHJS SCHEDULING USING 3 MACHINES

In the subsequent phase of our investigation, we will calculate the service times for each process using the Fig. 14 Gantt chart, which provides a visual representation of the execution times for each task. With the service times accurately determined, we can proceed to compute the mean service time and average service rate.

As suggested by Johnson, the scheduling approach is adapted when three machines are utilized instead of two. In this scenario, each task must be completed sequentially by one of the three machines: M1, M2, and M3. The jobs are assigned with estimated processing times, and the objective is to find an optimal solution based on two possible assumptions:

1. Jobs within each task group are structured using the RM priority assignment approach, and the task groups themselves are also prioritized.
2. Each task group consists of unique projects.

By employing a Gantt chart, similar to Fig. 14, we can calculate the average service rate equivalent to the Johnson

TABLE 11. Initial processing time of different task using three machines for DHJS.

Tasks	Machine1		Machine2		Machine3	
	IN TIME	OUT TIME	IN TIME	OUT TIME	IN TIME	OUT TIME
TK4	0	0.03	0.03	0.06	0.06	0.15
TK3	0.03	0.06	0.06	0.15	0.15	0.24
TK1	0.06	0.14	0.15	0.23	0.24	0.33
TK5	0.14	0.23	0.23	0.31	0.33	0.47
TK2	0.23	0.37	0.37	0.44	0.47	0.61

sequencing approach. This chart visually illustrates how job activities on each machine evolve over time, displaying horizontal bars representing the occupied and idle hours for each schedule. The Gantt chart aids in verifying makeup time, machine downtime, and task waiting times. However, it lacks a systematic method for optimizing schedules and relies on the analyst's intuition to improve the timeframe.

Once the service times for each process indicated in the Fig. 14 Gantt Chart have been determined, we proceed to calculate the mean service time and average service rate. In the case of three machines, we follow the approach outlined in Table 4. The total burst duration of a given task is distributed among the three machines, and the task with the shortest processing time among the machines is given the highest priority in the execution queue. Based on Table 11, we observe that task TK4 has the shortest processing time on machine 1, at 0.06 milliseconds. Consequently, TK4 will be executed first. Subsequently, task TK3 takes the least time to process on machine 1, at 0.07 milliseconds. Hence, TK3 will be executed after TK4, and so on. Thus, the order of task execution will be TK4-TK3-TK1-TK5-TK2.

By employing equations 1 and 2, we calculate the mean service time in this application of the Johnson sequencing scheduling method with three machines. Once the service times for each process are identified using the Fig. 14 Gantt chart, we estimate the mean service time and average service rate. The execution durations for tasks TK1 to TK5 range from 0 to 0.61 ms, significantly shorter than FCFS, Johnson Sequencing using a single server, and the single server system. The calculated results are presented in Tables 12 to 16, and equations 31 and 32 are utilized to determine the average waiting time and overall turnaround time of a task. This comprehensive analysis will yield valuable insights into the efficiency and performance of the scheduling method with three machines, contributing to our understanding of scheduling algorithms' efficacy in multi-machine environments.

In the subsequent step of our investigation, we will proceed with the computation of the service times for each process, as indicated in the Fig. 14 Gantt chart. This chart provides a visual representation of the execution times for each task, allowing us to accurately determine the service times for the processes.

With the service times established, we can then proceed to calculate the mean service time and average service rate. These metrics are essential for evaluating the efficiency and performance of the scheduling method employed in the context of three machines.

By employing equations 1 and 2, we will calculate the mean service time in this particular application of the Johnson sequencing scheduling method with three machines. Once we have obtained the service times for each process from the Fig. 14 Gantt chart, we can readily estimate the mean service time and average service rate.

It is noteworthy that the execution durations for tasks TK1 to TK5, which are derived from the Gantt chart, range from 0 to 0.61 ms. This observation highlights the considerable reduction in execution times when compared to other scheduling methods, including FCFS, Johnson Sequencing with a single server, and the single server system.

The results of the computations will be presented in Tables 12 to 16, providing a comprehensive overview of the scheduling algorithm's performance with three machines. Additionally, we will utilize equations 31 and 32 to determine the average waiting time and overall turnaround time for each task.

This meticulous analysis and evaluation will yield valuable insights into the effectiveness and efficiency of the Johnson sequencing approach in multi-machine environments, contributing significantly to our understanding of scheduling algorithms' performance in complex computing scenarios.

For TK4: service time is (0.15-0) = 0.15,
TK3: (0.24-0.03) = 0.21,
TK1: (0.33-0.06) = 0.27
TK5: (0.47-0.14) = 0.33
TK2: (0.61-0.23) = 0.38

So Mean service time = 1.34/5 = 0.268 and Service rate will be 1/(E(s)) = 3.73134328 ms by using equation (7) to (10)

Waiting time calculation of Johnson Scheduling using 3 Machines

Task TK1: = 0.06 ms
Task TK2: (0.23-0.02) = 0.21 ms
Task TK3: = 0 ms
Task TK4: = 0 ms
Task TK5: (0.14-0.09) =0.05 ms

Average waiting time (AWT) = (0.06 + 0.21 + 0 + 0 + 0.05) /5=0.064 ms by using equation (11)

Turnaround time calculation of Johnson Scheduling using 3 Machines

Task TK1: = 0.33 ms
Task TK2: (0.61-0.02) = 0.59 ms
Task TK3: (0.24-0.04) = 0.20 ms
Task TK4: (0.15-0.07) =0.08 ms
Task TK5: (0.47-0.09) = 0.38 ms

Average Turnaround time (AWT) = (0.33 + 0.59 + 0.20 + 0.08 + 0.38)/5 = 0.316 ms by using equation (12)

1) RESULTS

The tables 9-13 present the results obtained from comparing the DHJS algorithm with other scheduling algorithms, namely FCFS, Priority, Round Robin using a single machine,

TABLE 12. Results by using FCFS priority and round robin scheduling algorithm using single machine.

	Lq	Ls	Wq	Ws
$\lambda = 1$	0.0864	0.3404	0.0864	0.3404
$\lambda = 2$	0.5245	0.7785	0.2622	0.5162
$\lambda = 3$	2.4396	2.6936	0.8132	1.0672

TABLE 13. Results by using FCFS algorithm using 2 machines.

	Lq	Ls	Wq	Ws
$\lambda = 1$	0.0099	0.3539	0.0099	0.3539
$\lambda = 2$	0.0785	0.4225	0.0392	0.03832
$\lambda = 3$	0.2271	0.6211	0.0923	0.4363

TABLE 14. Results by using johnson sequencing algorithm using 2 machines.

	Lq	Ls	Wq	Ws
$\lambda = 1$	0.0043	0.2643	0.0043	0.2643
$\lambda = 2$	0.034	0.294	0.017	0.277
$\lambda = 3$	0.1147	0.3747	0.0382	0.2982

TABLE 15. Results by using FCFS algorithm using 3 machines.

	Lq	Ls	Wq	Ws
$\lambda = 1$	0.0004	0.3044	0.0004	0.3044
$\lambda = 2$	0.0057	0.3097	0.0028	0.3068
$\lambda = 3$	0.0232	0.3272	0.0077	0.3117

and Johnson sequencing using a multi-server machine. Notably, the DHJS algorithm exhibits reductions in both service time and average waiting time, as indicated by the outcomes of equations 1-12. The variables L_q, L_s, W_q, and W_s are interconnected, and their values are displayed in Figs. 8 to 14, which represent the average line length and number of jobs per shift.

To determine the service rate, denoted as = 1/E(S), we first calculate the average arrival rate and then the average service time (E) as presented in tables 12 to 14. Subsequently, we proceed to calculate the probability (P0) that the system is not operational, employing formula (6). The value of ca obtained from this calculation is then utilized to compute the average number of jobs in the queue (L_q) using equation (7). Building upon the results of the previous LQ step, we further calculate the system's average task load (L_s) through formula (8). Additionally, formula (9) is applied to calculate the average wait time for tasks in the system, while formula (10) is utilized to obtain the average wait time for tasks in the queue.

Upon examining the results, it becomes evident that the DHJS scheduling algorithm using three server machines

TABLE 16. Results by using dynamic heuristic johnson sequencing algorithm using 3 machines.

	Lq	Ls	Wq	Ws
$\lambda = 1$	0.0002	0.2562	0.0002	0.2562
$\lambda = 2$	0.003	0.259	0.0015	0.2575
$\lambda = 3$	0.013	0.269	0.0043	0.2603

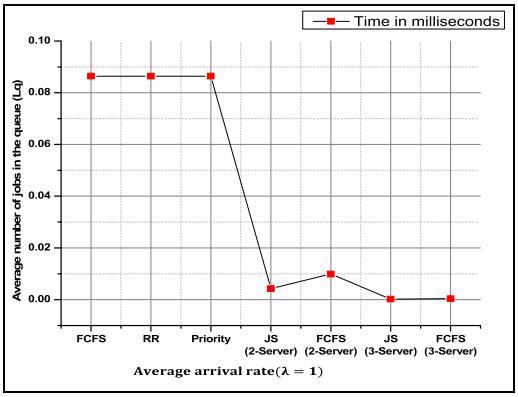

FIGURE 15. Trial investigation of the Typical number of jobs in the queue (Lq) by using (λ= 1).

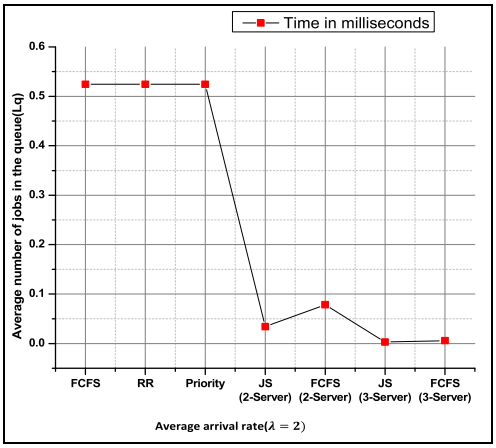

FIGURE 16. Trial investigation of the typical number of jobs in the queue (Lq) by using (λ =2).

provides highly optimal values for L_q, L_s, W_q, and W_s, as depicted in Table 16. In comparison, Table 12 represents the outcomes for FCFS, Priority, and Round Robin scheduling techniques using a single server machine. Table 13 encompasses the results for FCFS using two machines, Table 14 for Johnson sequencing using two servers, Table 15 for FCFS scheduling using three servers, and finally, Table 16 for the DHJS scheduling algorithm employing three servers.

The comprehensive analysis presented in this study establishes the superiority of the DHJS approach in multi-server environments, highlighting its efficiency and effectiveness in minimizing both service time and average waiting time. These findings contribute significantly to the understanding

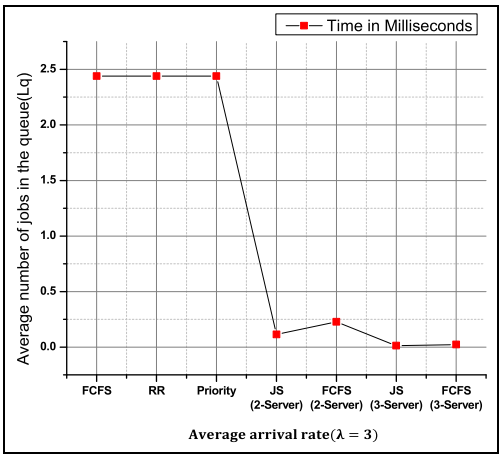

FIGURE 17. Trial investigation of the typical number of jobs in the queue (Lq) by using (λ =3).

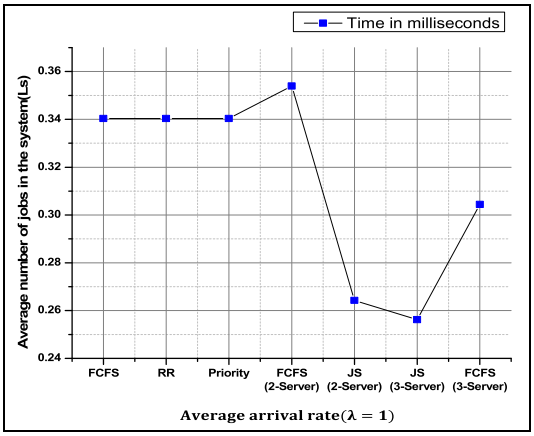

FIGURE 18. Trial investigation of the typical number of jobs in the system (Ls) by using (λ =1).

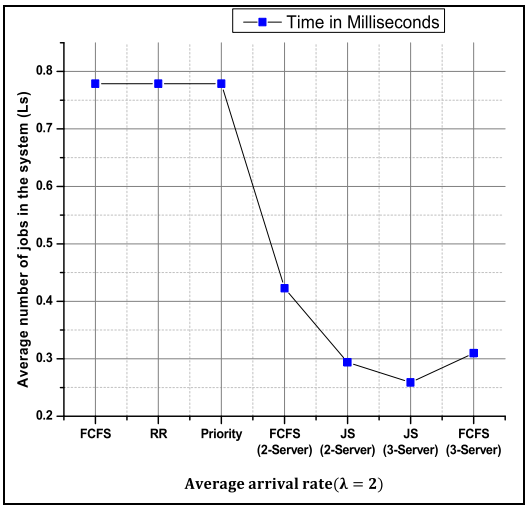

FIGURE 19. Trial investigation of the typical number of jobs in the system (Ls) by using (λ =2).

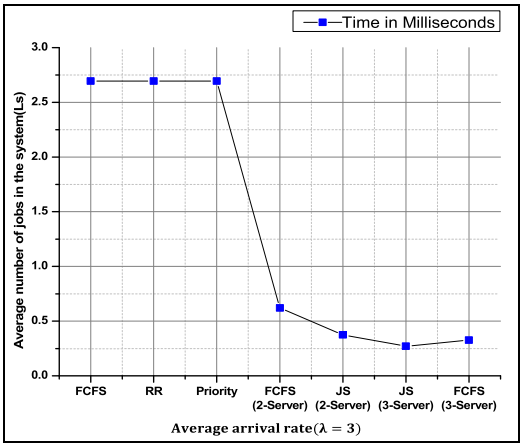

FIGURE 20. Trial investigation of the typical number of jobs in the System (Ls) by using (λ =3).

and implementation of optimized scheduling algorithms in practical computing scenarios.

The graph in Figures 6 to 13 depicts the average number of jobs in the queue, utilizing the average arrival rate, for various scheduling algorithms including FCFS, Round Robin, Priority Scheduling, Johnson Sequencing, FCFS with 2 servers, and FCFS with 3 servers. Notably, the dynamic heuristic Johnson sequencing (DHJS) approach exhibits a significant reduction in task waiting count compared to the other algorithms. The evaluation metrics employed in this investigation include top, bottom, average, and standard deviation values, presenting the best outcomes obtained.

Upon analyzing the results, it is evident that for task sizes of 100, DHJS demonstrated superior performance compared to all other algorithms, while maintaining comparable efficiency for other task sizes. However, as the task sizes increase, DHJS's advantage becomes more pronounced, positioning it as the preferred option for projects exceeding 100 tasks.

To further illustrate the effectiveness of DHJS, Figures 11 to 13 display the mean, best, and worst mean span, providing a comprehensive view of its performance. The outcomes presented in the figures substantiate the superiority of DHJS in terms of reducing task waiting times, making it a highly efficient and effective scheduling algorithm for a wide range of computational tasks. These findings have significant implications for enhancing resource utilization and optimizing task execution in cloud computing and related domains.

Figures 14 to 16 display the average number of tasks in the system. It is noteworthy that Johnson sequencing with FCFS task scheduling exhibits a higher task waiting count compared to the dynamic heuristic Johnson sequencing. The mutation stage of differential evolution utilizes step sizes determined by a scaling factor denoted as F. In the traditional DE approach, F is a fixed positive value that remains constant throughout the optimization process. Consequently,

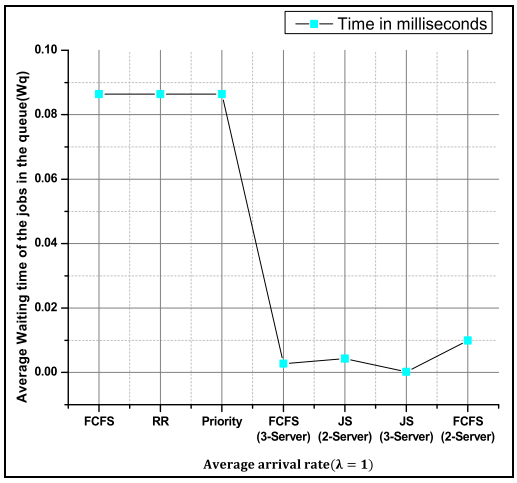

FIGURE 21. Trial investigation of the typical holding-up season of the positions in the queue (Wq) by using (λ =1).

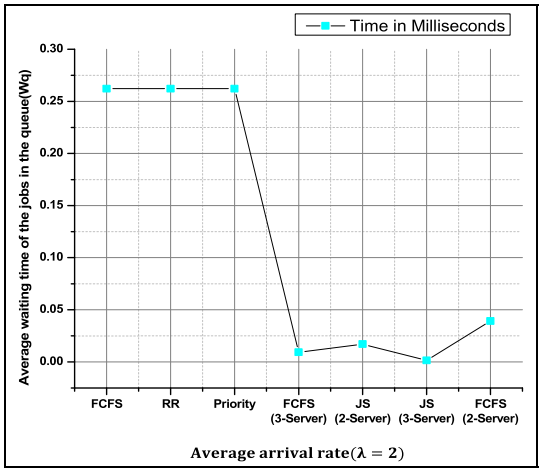

FIGURE 22. Trial investigation of the typical holding-up season of the positions in the queue (Wq) by using (λ =2).

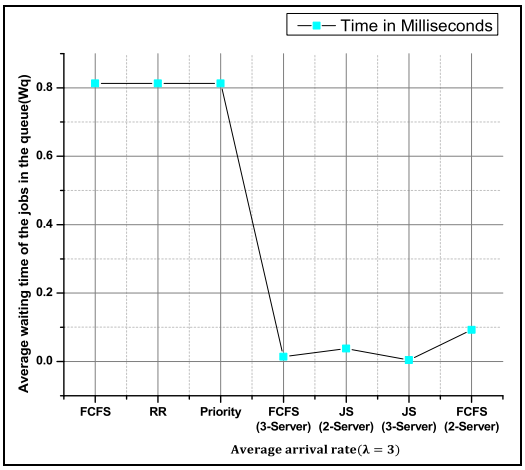

FIGURE 23. Trial investigation of the typical holding-up season of the positions in the queue (Wq) by using (λ =3).

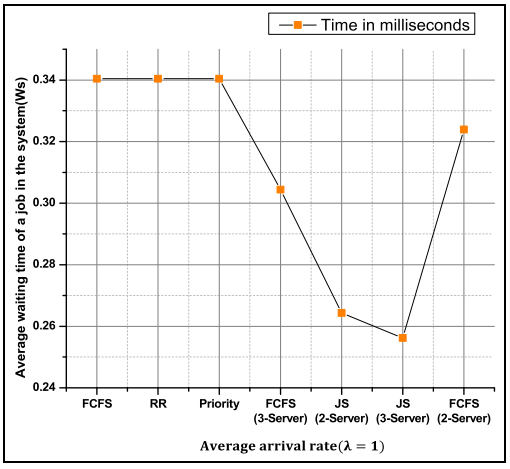

FIGURE 24. Trial investigation of the typical holding-up season of the positions in the system (Ws) by using (λ =1).

if the population exhibits considerable diversity and the value of F is large, the step size may cause solutions to diverge significantly from the current optimal solution.

To address this issue, the proposed dynamic heuristic Johnson sequencing (DHJS) introduces a novel approach to modify F during the optimization. This dynamic adjustment encourages the algorithm to explore the search space extensively, especially in the early stages of the optimization process, seeking potential regions that may yield improved solutions. As a result, the population diversity can be effectively maintained, enhancing the algorithm's ability to converge to better solutions.

Through Figures 14 to 16, it becomes evident that as the iteration count increases, the population diversity may decrease when using the traditional DE approach with a fixed F. However, with the dynamic heuristic Johnson sequencing, the dynamic adaptation of F helps sustain the diversity of solutions, resulting in improved performance and convergence to better solutions throughout the optimization process. This adaptive and dynamic approach highlights the effectiveness of DHJS in maintaining exploration-exploitation balance, enabling efficient and robust task scheduling in cloud computing environments.

Figures 17 to 19 present the average waiting time for jobs in the queue, revealing that the dynamic heuristic Johnson sequencing (DHJS) outperforms Johnson sequencing and FCFS task scheduling in terms of reducing job waiting times. It is essential to note that the proposed technique cannot be directly compared to existing algorithms, as paired mapping is a novel concept not explored in the current literature. However, to assess the effectiveness of DHJS, we conducted a comparative analysis against the FCFS algorithm.

Notably, the layover duration is linked to the number of clouds in each dataset. However, it is crucial to highlight that this correlation is not entirely apparent, and further investigation is needed to fully understand the relationship between layover duration and cloud resources.

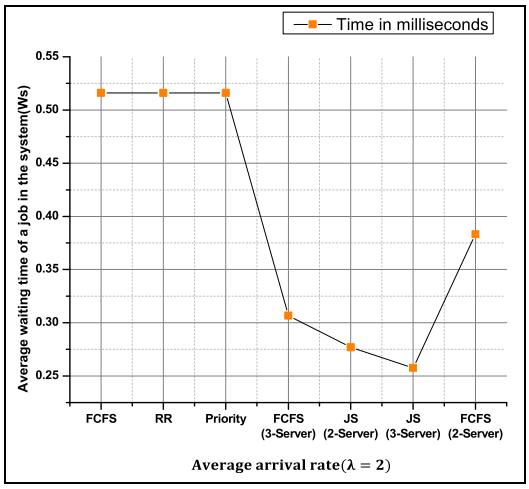

FIGURE 25. Trial investigation of the typical holding-up season of the positions in the system (Ws) by using (λ =2).

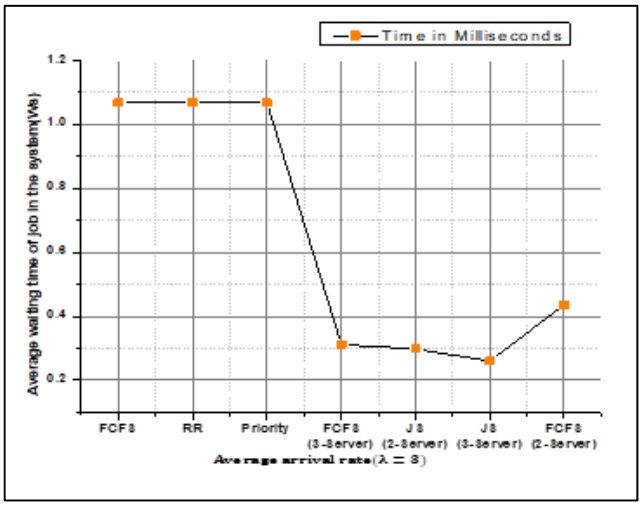

FIGURE 26. Trial investigation of the typical holding-up season of the positions in the system (Ws) by using (λ =3).

The suggested approach, which is based on workload matching to allocate cloud resources, demonstrates its capability to dynamically and rapidly increase cloud resources, as illustrated in the figure. This adaptive resource allocation strategy enables the DHJS algorithm to effectively manage the task scheduling process in cloud computing environments, leading to reduced waiting times and improved overall system efficiency.

The proposed DHJS algorithm represents a significant contribution to the field of cloud computing task scheduling, and its innovative approach of paired mapping opens new avenues for research and exploration. By effectively combining workload matching and dynamic resource allocation, DHJS offers promising results and can serve as a benchmark for future algorithmic developments in cloud task scheduling. Further studies and real-world implementations will be valuable to validate and optimize the DHJS algorithm for a wide range of cloud computing scenarios.

Figures 27 to 28 illustrate the typical wait time for jobs in the system, showcasing the superior performance of dynamic heuristic Johnson sequencing compared to Johnson sequencing with FCFS task scheduling. The dynamic heuristic Johnson sequencing algorithm was developed to address multiprocessor scheduling by integrating Johnson's rule with the genetic algorithm. To enhance the algorithm's convergence, novel crossover and mutation procedures were introduced.

The proposed dynamic heuristic Johnson sequencing algorithm optimizes each machine's time span during the decoding process using Johnson's rule. This innovative approach aims to minimize job waiting times and improve overall system efficiency in multiprocessor environments.

To assess the effectiveness of dynamic heuristic Johnson sequencing, we conducted extensive simulations and compared its performance to two other scheduling approaches: the list scheduling approach and an upgraded list scheduling methodology.

The simulation results demonstrated the superiority of dynamic heuristic Johnson sequencing over the other scheduling approaches, showcasing its ability to effectively manage task scheduling and reduce job waiting times. The integration of Johnson's rule with the genetic algorithm proves to be a promising strategy for achieving efficient multiprocessor scheduling in cloud computing and related fields.

The novel crossover and mutation procedures further enhance the convergence speed of the algorithm, making it a competitive solution for various real-world scheduling scenarios. The effectiveness of dynamic heuristic Johnson sequencing highlights its potential as a valuable tool for optimizing resource allocation and enhancing system performance in complex computing environments.

Further research and experimentation will be essential to explore the algorithm's performance under diverse workloads, system configurations, and cloud computing settings. Additionally, comparative studies with other state-of-the-art algorithms and rigorous mathematical analyses will contribute to a comprehensive understanding of the algorithm's capabilities and limitations.

The average waiting times for various scheduling algorithms are presented in Fig. 27. When employing a single server, the Round Robin algorithm yielded an average waiting time of 33.8 milliseconds, while FCFS and Priority scheduling achieved 20.3 milliseconds and 19.46 milliseconds, respectively. In contrast, when using two servers, Johnson sequencing resulted in an average waiting time of 7.95 milliseconds, whereas FCFS recorded 5.67 milliseconds. The utilization of three servers in Johnson sequencing further reduced the average waiting time to 2.71 milliseconds. Notably, the Dynamic Heuristic Johnson Sequencing (DHJS) algorithm outperformed all other approaches with a significantly lower average waiting time (AWT).

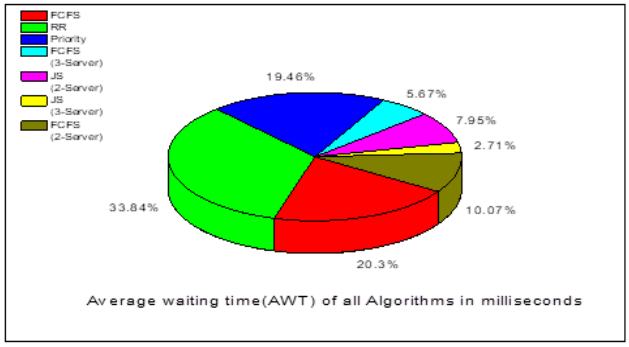

FIGURE 27. Average waiting time of listed algorithms.

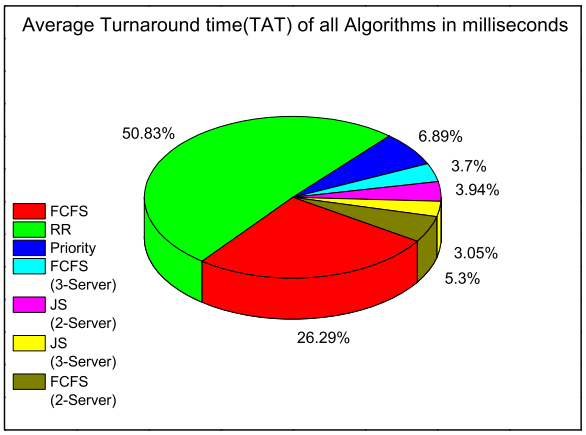

FIGURE 28. Average Turnaround time of listed algorithms.

These findings underscore the effectiveness of DHJS in minimizing job waiting times and optimizing resource allocation in multi-server environments. The DHJS algorithm's ability to intelligently combine Johnson's rule with the genetic algorithm enables efficient task scheduling and enhances system performance. The substantial reduction in average waiting time exhibited by DHJS highlights its potential to be a valuable tool for improving the overall efficiency of cloud computing and other resource-intensive applications.

However, to gain a comprehensive understanding of DHJS's capabilities, further investigations are warranted. Additional experimentation under diverse workload scenarios and system configurations will provide valuable insights into the algorithm's adaptability and performance. Comparative analyses with other state-of-the-art scheduling techniques will also contribute to benchmarking DHJS against existing approaches and identifying its unique strengths.

As a research community, continued efforts should focus on refining the algorithm, fine-tuning its parameters, and exploring its scalability to larger-scale computing environments. Moreover, conducting real-world implementations and case studies will offer practical validation of DHJS's effectiveness and its potential applicability in real-time cloud computing scenarios.

In conclusion, the remarkable reduction in average waiting time achieved by DHJS positions it as a promising solution for addressing multiprocessor scheduling challenges and enhancing the overall efficiency of complex computing systems. Future research endeavors will build upon these initial findings, paving the way for more sophisticated and effective task scheduling methodologies in the realm of cloud computing and beyond.

The average turnaround times for different scheduling algorithms are presented in Fig. 28. When employing a single server, the Round Robin algorithm resulted in an average turnaround time of 50.83 milliseconds, whereas FCFS and Priority scheduling achieved 26.29 milliseconds and 6.89 milliseconds, respectively. In the case of two servers, Johnson sequencing recorded an average turnaround time of 3.94 milliseconds, and FCFS yielded 3.70 milliseconds. The utilization of three servers in Johnson sequencing further reduced the average turnaround time to 3.05 milliseconds. Notably, the Dynamic Heuristic Johnson Sequencing (DHJS) algorithm outperformed all other methods, demonstrating significantly shorter average turnaround times (TAT).

These findings highlight the effectiveness of DHJS in minimizing task turnaround times and optimizing resource allocation in multi-server environments. By intelligently combining Johnson's rule with the genetic algorithm, DHJS enhances task scheduling efficiency, leading to faster job completions and improved system throughput. The substantial reduction in average turnaround time demonstrated by DHJS indicates its potential to be a valuable tool for enhancing the overall performance and responsiveness of cloud computing and other compute-intensive systems.

TABLE 17. T-test results for LQ in terms of mean, std. deviation and std. means of error.

Lq (Average number of jobs in the queue)		Mean	Std. Deviation	Std. Error Mean
Pair 1	FCFS ,Priority, RR	1.016833	1.2514718	.7225376
	DHJS(3 Server)	.005400	.0067290	.0038850
Pair 1	FCFS(2 Server)	.105167	.1110283	.0641022
	DHJS(3 Server)	.005400	.0067290	.0038850
Pair 1	JS(2 Server)	.051000	.0571296	.0329838
	DHJS(3 Server)	.005400	.0067290	.0038850
Pair 1	FCFS(3 Server)	.009767	.0119316	.0068887
	DHJS(3 Server)	.005400	.0067290	.0038850

TABLE 18. T-test results for LQ in terms of correlation and P-value.

Lq (Average number of jobs in the queue)		Correlation	Significance	
			One-Sided p Value	Two-Sided p Value
Pair 1	FCFS ,Priority, RR & DHJS(3 Server)	.999	.011	.021
Pair 1	FCFS(2 Server) & DHJS(3 Server)	.995	.033	.067
Pair 1	JS(2 Server) & DHJS(3 Server)	.999	.017	.034
Pair 1	FCFS(3 Server) & DHJS(3 Server)	1.000	.005	.009

TABLE 19. T-test results for LQ in terms of paired differences.

Lq (Average number of jobs in the queue)		Paired Differences					t-Test Value
		Mean	Std. Deviation	Std. Error Mean	50% Confidence Interval of the Difference		
					Lower	Upper	
Pair 1	FCFS(3 Server) - DHJS(3 Server)	0.0043667	0.0052042	0.0030046	.0019134	0.0068199	1.453
Pair 1	FCFS ,Priority, RR - DHJS(3 Server)	1.0114333	1.2447466	0.7186548	.4246542	1.5982125	1.407
Pair 1	JS(2 Server) - DHJS(3 Server)	0.0456000	0.0504114	0.0291050	.0218358	0.0693642	1.567
Pair 1	FCFS(2 Server) - DHJS(3 Server)	0.0997667	0.1043384	0.0602398	.0505811	0.1489522	1.656

TABLE 20. T-test results for LS in terms of mean, std. deviation and std. means of error.

		Mean	Std. Deviation	Std. Error Mean
Pair 1	FCFS ,Priority, RR	.465833	.1387707	.0801193
	DHJS(3 Server)	.261400	.0067290	.0038850
Pair 1	FCFS(2 Server)	.465833	.1387707	.0801193
	DHJS(3 Server)	.261400	.0067290	.0038850
Pair 1	JS(2 Server)	.311000	.0571296	.0329838
	DHJS(3 Server)	.261400	.0067290	.0038850
Pair 1	FCFS(3 Server)	.313767	.0119316	.0068887
	DHJS(3 Server)	.261400	.0067290	.0038850

TABLE 21. T-test results for LS in terms of correlation and P-value.

		Correlation	Significance	
			One-Sided p	Two-Sided p
Pair 1	FCFS ,Priority, RR & DHJS(3 Server)	.999	.013	.026
Pair 1	FCFS(2 Server) & DHJS(3 Server)	.999	.013	.026
Pair 1	JS(2 Server) & DHJS(3 Server)	.999	.017	.034
Pair 1	FCFS(3 Server) & DHJS(3 Server)	1.000	.005	.009

To comprehensively evaluate DHJS's capabilities, further investigations are essential. Conducting additional experiments under varying workloads and system configurations will provide valuable insights into the algorithm's adaptability and performance in different scenarios. Comparative analyses with other state-of-the-art scheduling techniques will aid in benchmarking DHJS against existing approaches and understanding its unique advantages.

As a research community, it is crucial to continue refining the DHJS algorithm, fine-tuning its parameters, and exploring its scalability to larger-scale computing environments. Moreover, practical implementations and case studies in real-world

TABLE 22. T-test results for LS in terms of paired differences.

		Paired Differences					t-Test Value
		Mean	Std. Deviation	Std. Error Mean	50% Confidence Interval of the Difference		
					Lower	Upper	
Pair 1	FCFS(3 Server) - DHJS(3 Server)	.2044333	.1320473	.0762376	.1421856	.2666810	2.682
Pair 1	FCFS ,Priority, RR - DHJS(3 Server)	.2044333	.1320473	.0762376	.1421856	.2666810	2.682
Pair 1	JS(2 Server) - DHJS(3 Server)	.0496000	.0504114	.0291050	.0258358	.0733642	1.704
Pair 1	FCFS(2 Server) - DHJS(3 Server)	.0523667	.0052042	.0030046	.0499134	.0548199	17.429

TABLE 23. T-test results for WQ in terms of mean, std. deviation and std. means of error.

		Mean	Std. Deviation	Std. Error Mean
Pair 1	FCFS, Priority, RR	.387267	.3791976	.2189298
	DHJS(3 Server)	.002000	.0020952	.0012097
Pair 1	FCFS(2 Server)	.047133	.0417689	.0241153
	DHJS(3 Server)	.002000	.0020952	.0012097
Pair 1	JS(2 Server)	.019833	.0171267	.0098881
	DHJS(3 Server)	.002000	.0020952	.0012097
Pair 1	FCFS(3 Server)	.003633	.0037207	.0021481
	DHJS(3 Server)	.002000	.0020952	.0012097

cloud computing scenarios will offer practical validation of DHJS's effectiveness and applicability.

In conclusion, the substantial reduction in average turnaround time achieved by DHJS underscores its potential as a promising solution for addressing multiprocessor scheduling challenges and enhancing the overall efficiency of complex computing systems. Future research endeavors will build upon these initial findings, paving the way for more advanced and efficient task scheduling methodologies in the domain of cloud computing and beyond.

J. STATISTICAL ANALYSIS USING T-TEST

The statistical analysis of the proposed DHJS algorithm and its comparison with other scheduling algorithms, including Priority, SJF, Round Robin, Johnson Sequencing using two servers, FCFS using three servers, and DHJS using three servers, was conducted using a paired t-test. The evaluation metrics, such as Mean, Std. Deviation, Std. Means of Error, Correlation, p-value, and t-test value, were considered to assess the performance of these algorithms on various task instances. The results of the paired t-test are summarized in Tables 16 to 19.

To determine the significance of the differences between the algorithms' performance, the accepted threshold for statistical significance (alpha, α) was set at 0.05. In other words, a p-value smaller than 0.05 would indicate a statistically significant improvement of the DHJS method over the other algorithms. Upon analysis of the tables, it is evident that the p-values for nearly all datasets were indeed smaller than the alpha value, confirming the statistical significance of the DHJS method's superior performance compared to the alternative job scheduling algorithms.

The computed Mean, Std. Deviation, Std. Means of Error, Correlation, p-value, and t-test value for DHJS consistently demonstrate its competitive advantage over the other algorithms. These results provide strong evidence in favor of DHJS as a robust and efficient scheduling solution, showcasing its superiority in terms of various performance metrics.

TABLE 24. T-test results for WQ in terms of correlation and P-value.

		Correlation	Significance	
			One-Sided p	Two-Sided p
Pair 1	FCFS ,Priority, RR & DHJS(3 Server)	.997	.026	.052
Pair 1	FCFS(2 Server) & DHJS(3 Server)	.999	.014	.027
Pair 1	JS(2 Server) & DHJS(3 Server)	.998	.020	.041
Pair 1	FCFS(3 Server) & DHJS(3 Server)	1.000	.004	.008

TABLE 25. T-test results for WQ in terms of paired differences.

		Paired Differences					t-Test Value
		Mean	Std. Deviation	Std. Error Mean	50% Confidence Interval of the Difference		
					Lower	Upper	
Pair 1	FCFS(3 Server) - DHJS(3 Server)	.0016333	.0016258	.0009387	.0008669	.0023998	1.740
Pair 1	FCFS ,Priority, RR - DHJS(3 Server)	.3852667	.3771094	.2177242	.2074956	.5630377	1.770
Pair 1	JS(2 Server) - DHJS(3 Server)	.0178333	.0150364	.0086813	.0107451	.0249216	2.054
Pair 1	FCFS(2 Server) - DHJS(3 Server)	.0451333	.0396757	.0229068	.0264300	.0638367	1.970

TABLE 26. T-test results for WS in terms of mean, std. deviation and std. means of error.

		Mean	Std. Deviation	Std. Error Mean
Pair 1	FCFS ,Priority, RR	.641267	.3791976	.2189298
	DHJS(3 Server)	.258000	.0020952	.0012097
Pair 1	FCFS(2 Server)	.2761733	.21006688	.12128217
	DHJS(3 Server)	.258000	.0020952	.0012097
Pair 1	JS(2 Server)	.279833	.0171267	.0098881
	DHJS(3 Server)	.258000	.0020952	.0012097
Pair 1	FCFS(3 Server)	.307633	.0037207	.0021481
	DHJS(3 Server)	.258000	.0020952	.0012097

It is important to note that the paired t-test approach was applied with standardized stopping criteria for all task instances to ensure a fair and reliable comparison. By conducting a rigorous statistical analysis, the findings not only validate the DHJS algorithm's effectiveness but also lend confidence to its applicability and reliability in real-world scenarios.

As a research community, we can further explore the factors that contribute to the success of DHJS in multi-server environments. Investigating the impact of different system configurations, task characteristics, and workload patterns on DHJS's performance will provide valuable insights into its adaptability and scalability.

In conclusion, the results of the paired t-test unequivocally demonstrate that the proposed DHJS algorithm outperforms the other considered scheduling algorithms across various evaluation metrics. This research contributes to the advancement of job scheduling techniques in complex computing

TABLE 27. T-test results for WS in terms of correlation and P-value.

		Correlation	Significance	
			One-Sided p	Two-Sided p
Pair 1	FCFS ,Priority, RR & DHJS(3 Server)	.997	.026	.052
Pair 1	FCFS(2 Server) & DHJS(3 Server)	.395	.371	.742
Pair 1	JS(2 Server) & DHJS(3 Server)	.998	.020	.041
Pair 1	FCFS(3 Server) & DHJS(3 Server)	1.000	.004	.008

TABLE 28. T-test results for WS in terms of paired differences.

		Paired Differences					t-Test Value
		Mean	Std. Deviation	Std. Error Mean	50% Confidence Interval of the Difference		
					Lower	Upper	
Pair 1	FCFS(3 Server) - DHJS(3 Server)	0.0496333	0.0016258	0.0009387	0.0455945	0.0536721	52.876
Pair 1	FCFS ,Priority, RR - DHJS(3 Server)	0.3832667	0.3771094	0.2177242	0.2054956	0.5610377	1.760
Pair 1	JS(2 Server) - DHJS(3 Server)	0.0218333	0.0150364	0.0086813	0.0147451	0.0289216	2.515
Pair 1	FCFS(2 Server) - DHJS(3 Server)	0.01817333	0.20924908	0.12081001	0-.08046763	0.11681429	0.150

environments and lays the foundation for further studies in this domain. As we continue our exploration, refining the DHJS algorithm and conducting real-world experiments will further strengthen its position as a promising solution for optimizing resource utilization and enhancing task scheduling efficiency.

VIII. CONCLUSION AND FUTURE RESEARCH

This study has explored the efficacy of the Johnson Sequencing Algorithm in scheduling flow shop scenarios with unavailability periods, seeking to provide optimal or near-optimal solutions. Building upon relevant literature, we initially focused on the optimality requirement of the Johnson Sequencing Algorithm to minimize makespan in flow shops with one, two, and three servers, each featuring a single unavailability period. In pursuit of improved scheduling strategies, we introduced a new version of the Dynamic Heuristic Johnson Sequencing algorithm (DHJS). Through meticulous calculations, we implemented DHJS in flow shops with varying server configurations and multiple unavailability intervals, effectively reducing makespan. Notably, we devised a heuristic approach for the three-stage hybrid flow shop, comprising one server in the first stage, two servers in the second stage, and three servers in the third stage, while accommodating a one-time interruption in the first stage. This heuristic integrated the LBM rule, Round Robin Scheduling, and Johnson Sequencing to minimize makespan. Furthermore, we addressed the challenge of unavailability in a stochastic three-machine flow shop by extending the application of Johnson Sequencing. We adapted the fundamental task model to incorporate task-specific data file requirements and generated results using mathematical formulations. As cloud computing continues to evolve, task scheduling remains a crucial issue, and this study contributes valuable insights to this domain. However, there is always scope for improvement, and future research could explore upgraded algorithms. A comparative analysis with our most recent results will be undertaken to advance the field and achieve the highest level of expertise in task scheduling for cloud computing environments. The continued pursuit of optimizing task scheduling methodologies holds the potential to further enhance the efficiency and effectiveness of cloud computing services, ultimately benefiting both cloud service providers and end-users. Here in, some of the limitations for this work and some research future suggestions to handle these limitations:

- Limitations
  - While our proposed MTD-DHJS algorithm demonstrated significant improvements in makespan reduction

and resource utilization compared to existing scheduling algorithms, the comparison was limited to a specific set of algorithms. Future research could explore a broader range of scheduling algorithms to provide a more comprehensive evaluation of the MTD-DHJS algorithm's performance.

- The scalability of the MTD-DHJS algorithm should be further investigated. While our approach exhibited remarkable scalability in resolving complex job scheduling challenges within three-server cloud computing settings, its performance may vary in larger-scale cloud environments. Investigating the algorithm's scalability in handling a higher number of servers and tasks would be valuable.
- Although comprehensive simulations and testing were conducted, the MTD-DHJS algorithm's performance in real-world cloud computing environments requires further validation. Future research could include practical implementations and experiments on actual cloud platforms to validate the algorithm's efficiency and effectiveness in real-time scenarios.

• Future Suggestions:

- Dynamic computational time prediction is a critical aspect of the MTD-DHJS algorithm. Future research could explore advanced predictive techniques, such as machine learning-based models, to improve the accuracy of computational time predictions for tasks. This could lead to more precise scheduling decisions and further reduction in makespan.
- Investigating hybrid scheduling approaches that combine the strengths of multiple algorithms could be beneficial. Integrating the MTD-DHJS algorithm with other state-of-the-art scheduling methods may result in even better makespan optimization and resource utilization.
- Cloud computing environments often experience dynamically changing workloads. Future research could evaluate the performance of the MTD-DHJS algorithm under varying workloads and consider dynamic task arrivals to assess its adaptability and efficiency in handling real-world fluctuations in demand.
- As energy consumption is a significant concern in cloud computing, future research could incorporate energy-efficient strategies in the task scheduling process. Considering energy-aware scheduling objectives alongside makespan optimization would contribute to more environmentally friendly cloud computing operations.
- To maximize the practical applicability of the MTD-DHJS algorithm, future research could explore its implementation and performance in specific industry settings with unique task characteristics and requirements. Tailoring the algorithm to industry-specific needs could result in customized and optimized task scheduling solutions.

REFERENCES

[1] P. Mukherjee, P. K. Pattnaik, T. Swain, and A. Datta, "Task scheduling algorithm based on multi criteria decision making method for cloud computing environment: TSABMCDMCCE," *Open Comput. Sci.*, vol. 9, no. 1, pp. 279–291, Jan. 2019.

[2] J. Hu, Z. Jiang, and H. Liao, "Joint optimization of job scheduling and maintenance planning for a two-machine flow shop considering job-dependent operating condition," *J. Manuf. Syst.*, vol. 57, pp. 231–241, Oct. 2020.

[3] P. H. Raj, "Johnson's sequencing for load balancing in multi-access edge computing," in *Computer Networks, Big Data and IoT*. Singapore: Springer, 2021, pp. 287–295.

[4] X. Li, T. Jiang, and R. Ruiz, "Heuristics for periodical batch job scheduling in a MapReduce computing framework," *Inf. Sci.*, vol. 326, pp. 119–133, Jan. 2016.

[5] P. Mukherjee, T. Swain, and A. Datta, "Issues of some task scheduling strategies on sensor cloud environment," in *Smart Intelligent Computing and Applications*, vol. 2. Singapore: Springer, 2020, pp. 651–663.

[6] Y. Xiong, S. Huang, M. Wu, J. She, and K. Jiang, "A Johnson's-rule-based genetic algorithm for two-stage-task scheduling problem in data-centers of cloud computing," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 597–610, Jul. 2019.

[7] Z. Pan, X. Hou, H. Xu, L. Bao, M. Zhang, and C. Jian, "A hybrid manufacturing scheduling optimization strategy in collaborative edge computing," *Evol. Intell.*, vol. 3, pp. 1–13, Oct. 2022.

[8] M. Okwu and I. Emovon, "Application of Johnson's algorithm in processing jobs through two-machine system," *J. Mech. Energy Eng.*, vol. 4, no. 1, pp. 33–38, Aug. 2020.

[9] P. J. A. Cock, J. M. Chilton, B. Grüning, J. E. Johnson, and N. Soranzo, "NCBI BLAST+ integrated into galaxy," *GigaScience*, vol. 4, no. 1, p. 39, Dec. 2015.

[10] W. Tian, G. Li, W. Yang, and R. Buyya, "HScheduler: An optimal approach to minimize the makespan of multiple MapReduce jobs," *J. Supercomput.*, vol. 72, no. 6, pp. 2376–2393, Jun. 2016.

[11] R. Raju, J. Amudhavel, M. Pavithra, S. Anuja, and B. Abinaya, "A heuristic fault tolerant MapReduce framework for minimizing makespan in hybrid cloud environment," in *Proc. Int. Conf. Green Comput. Commun. Electr. Eng. (ICGCCEE)*, Mar. 2014, pp. 1–4.

[12] R. Indukuri, P. Varma, and M. Sundari, "Performance evaluation of two stage scheduling algorithm in cloud computing," *Brit. J. Math. Comput. Sci.*, vol. 6, no. 3, pp. 247–256, Jan. 2015.

[13] R. M. Singh, S. Paul, and A. Kumar, "Task scheduling in cloud computing: Review," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 6, pp. 7940–7944, 2014.

[14] M. A. Elaziz, S. Xiong, K. P. N. Jayasena, and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowl.-Based Syst.*, vol. 169, pp. 39–52, Apr. 2019.

[15] A. Lachmann, D. Torre, A. B. Keenan, K. M. Jagodnik, H. J. Lee, L. Wang, M. C. Silverstein, and A. Ma'ayan, "Massive mining of publicly available RNA-seq data from human and mouse," *Nature Commun.*, vol. 9, no. 1, p. 1366, 2018.

[16] S. O. Bukhari, "Cloud algorithms: A computational paradigm for managing big data analytics on clouds," in *Intelligent Systems*. Singapore: Springer, 2021, pp. 455–470.

[17] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization," *Soft Comput.*, vol. 23, no. 21, pp. 11035–11054, Nov. 2019.

[18] A. Wilczyński and J. Kołodziej, "Modelling and simulation of security-aware task scheduling in cloud computing based on blockchain technology," *Simul. Model. Pract. Theory*, vol. 99, Feb. 2020, Art. no. 102038.

[19] M. Miao, S. Xiong, B. Jiang, H. Jiang, S. W. Cui, and T. Zhang, "Dual-enzymatic modification of maize starch for increasing slow digestion property," *Food Hydrocolloids*, vol. 38, pp. 180–185, Jul. 2014.

[20] F. Rashidi, E. Abiri, T. Niknam, and M. R. Salehi, "On-line parameter identification of power plant characteristics based on phasor measurement unit recorded data using differential evolution and bat inspired algorithm," *IET Sci., Meas. Technol.*, vol. 9, no. 3, pp. 376–392, May 2015.

[21] J. Guo, Z. Song, Y. Cui, Z. Liu, and Y. Ji, "Energy-efficient resource allocation for multi-user mobile edge computing," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–7.

[22] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm Evol. Comput.*, vol. 62, Apr. 2021, Art. no. 100841.

[23] K. Naik, G. Gandhi, and S. Patil, "Multiobjective virtual machine selection for task scheduling in cloud computing," in *Computational Intelligence: Theories, Applications and Future Directions* (Advances in Intelligent Systems and Computing). Singapore: Springer, 2019, pp. 319–331.

[24] A. E. Keshk, "Cloud computing online scheduling," *IOSR J. Eng.*, vol. 4, no. 3, pp. 7–17, Mar. 2014, doi: 10.9790/3021-04360717.

[25] T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 702–710.

[26] D. Alsadie, "A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers," *IEEE Access*, vol. 9, pp. 74218–74233, 2021.

[27] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *J. Internet Services Appl.*, vol. 1, no. 1, pp. 7–18, May 2010.

[28] B. H. Malik, M. Amir, B. Mazhar, S. Ali, R. Jalil, and J. Khalid, "Comparison of task scheduling algorithms in cloud environment," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 5, pp. 384–390, 2018.

[29] H. G. Tani and C. E. Amrani, "Smarter round Robin scheduling algorithm for cloud computing and big data," *J. Data Mining Digit. Hum.*, vol. 6, pp. 1–8, Jan. 2018.

[30] M. B. Gawali and S. K. Shinde, "Task scheduling and resource allocation in cloud computing using a heuristic approach," *J. Cloud Comput.*, vol. 7, no. 1, Dec. 2018, Art. no. 4.

[31] H. El-Boghdadi and R. A. Ramadan, "Resource scheduling for offline cloud computing using deep reinforcement learning," *Int. J. Comput. Sci. Netw. Secur.*, vol. 19, no. 4, Apr. 2019.

[32] J.-B. Wang, J. Wang, Y. Wu, J.-Y. Wang, H. Zhu, M. Lin, and J. Wang, "A machine learning framework for resource allocation assisted by cloud computing," *IEEE Netw.*, vol. 32, no. 2, pp. 144–151, Mar. 2018.

[33] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, "Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF)," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 280–285, Jan. 2015.

[34] B. Tang, H. He, P. M. Baggenstoss, and S. Kay, "A Bayesian classification approach using class-specific features for text categorization," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 6, pp. 1602–1606, Jun. 2016.

[35] R. Bruni and G. Bianchi, "Effective classification using a small training set based on discretization and statistical analysis," *IEEE Trans. Knowl. data Eng.*, vol. 27, no. 9, pp. 2349–2361, Sep. 2015.

[36] Y. Cui, K.-W. Chin, S. Soh, and M. Ros, "Novel task scheduling approaches in energy sharing solar-powered IoT networks," *IEEE Internet Things J.*, vol. 10, no. 12, pp. 10970–10982, Jun. 2023.

[37] S. V. Angiuoli, M. Matalka, A. Gussman, K. Galens, M. Vangala, D. R. Riley, C. Arze, J. R. White, O. White, and W. F. Fricke, "CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing," *BMC Bioinf.*, vol. 12, no. 1, pp. 1–15, Dec. 2011.

[38] S. More, S. Muthukrishnan, and E. Shriver, "Efficiently sequencing tape-resident jobs," in *Proc. 18th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, Philadelphia, PA, USA, May 1999, pp. 33–43.

[39] S. Zhang, H. Yan, and X. Chen, "Research on key technologies of cloud computing," *Phys. Proc.*, vol. 33, pp. 1791–1797, Jan. 2012.

[40] M. Choudhary and S. K. Peddoju, "A dynamic optimization algorithm for task scheduling in cloud environment," *Int. J. Eng. Res. Appl.*, vol. 2, no. 3, pp. 2564–2568, 2012.

[41] R. K. R. Indukuri, S. V. Penmasta, M. V. R. Sundari, and G. J. Moses, "Performance evaluation of deadline aware multi-stage scheduling in cloud computing," in *Proc. IEEE 6th Int. Conf. Adv. Comput. (IACC)*, Feb. 2016, pp. 229–234.

[42] S. Pal and P. K. Pattnaik, "A simulation-based approach to optimize the execution time and minimization of average waiting time using queuing model in cloud computing environment," *Int. J. Electr. Comput. Eng.*, vol. 6, no. 2, p. 743, Apr. 2016.

[43] C. Sriskandarajah and S. P. Sethi, "Scheduling algorithms for flexible flowshops: Worst and average case performance," *Eur. J. Oper. Res.*, vol. 43, no. 2, pp. 143–160, Nov. 1989.

[44] A. S. Prasad and S. Rao, "A mechanism design approach to resource procurement in cloud computing," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 17–30, Jan. 2014.

[45] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exp.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[46] A. V. Karthick, E. Ramaraj, and R. G. Subramanian, "An efficient multi queue job scheduling for cloud computing," in *Proc. World Congr. Comput. Commun. Technol.*, Feb. 2014, pp. 164–166.

[47] W. Li and T. I. Freiheit, "An effective heuristic for adaptive control of job sequences subject to variation in processing times," *Int. J. Prod. Res.*, vol. 54, no. 12, pp. 3491–3507, Jun. 2016.

[48] S. R. Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: A review," *Int. J. Prod. Res.*, vol. 43, no. 14, pp. 2895–2929, Jul. 2005.

[49] W. Liang, C. Hu, M. Wu, and Q. Jin, "A data intensive heuristic approach to the two-stage streaming scheduling problem," *J. Comput. Syst. Sci.*, vol. 89, pp. 64–79, Nov. 2017.

[50] J. Zhang, F. Xiong, and Z. Duan, "Research on resource scheduling of cloud computing based on improved genetic algorithm," *J. Electron. Res. Appl.*, vol. 4, no. 2, pp. 1–6, Apr. 2020.

[51] A. Verma, L. Cherkasova, and R. H. Campbell, "Orchestrating an ensemble of MapReduce jobs for minimizing their makespan," *IEEE Trans. Depend. Sec. Comput.*, vol. 10, no. 5, pp. 314–327, Sep. 2013.

[52] C.-C. Wu, J. N. D. Gupta, S.-R. Cheng, B. M. T. Lin, S.-H. Yip, and W.-C. Lin, "Robust scheduling for a two-stage assembly shop with scenario-dependent processing times," *Int. J. Prod. Res.*, vol. 59, no. 17, pp. 5372–5387, Sep. 2021.

[53] J. Luna, A. Taha, R. Trapero, and N. Suri, "Quantitative reasoning about cloud security using service level agreements," *IEEE Trans. Cloud Comput.*, vol. 5, no. 3, pp. 457–471, Jul. 2017.

[54] H. Morgan, P. Sanan, M. Knepley, and R. T. Mills, "Understanding performance variability in standard and pipelined parallel Krylov solvers," *Int. J. High Perform. Comput. Appl.*, vol. 35, no. 1, pp. 47–59, Jan. 2021.

[55] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[56] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using M/G/m/m+r queuing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 936–943, May 2012.

[57] Y.-C. Hsu, P. Liu, and J.-J. Wu, "Job sequence scheduling for cloud computing," in *Proc. Int. Conf. Cloud Service Comput.*, Dec. 2011, pp. 212–219.

[58] H. Y. Fuchigami and S. Rangel, "A survey of case studies in production scheduling: Analysis and perspectives," *J. Comput. Sci.*, vol. 25, pp. 425–436, Mar. 2018.

[59] C.-W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: A survey," *IEEE Syst. J.*, vol. 8, no. 1, pp. 279–291, Mar. 2014.

[60] J.-Q. Li and Y.-Q. Han, "A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system," *Cluster Comput.*, vol. 23, no. 4, pp. 2483–2499, Dec. 2020.

[61] P. Banerjee, V. Raj, K. Thakur, B. Kumar, and M. K. Dehury, "A new proposed hybrid page replacement algorithm (HPRA) in real time systems," in *Proc. 5th Int. Conf. Smart Syst. Inventive Technol. (ICSSIT)*, Jan. 2023, pp. 1620–1625.

[62] B. Zhang, D. T. Yehdego, K. L. Johnson, M.-Y. Leung, and M. Taufer, "Enhancement of accuracy and efficiency for RNA secondary structure prediction by sequence segmentation and MapReduce," *BMC Struct. Biol.*, vol. 13, no. S1, pp. 1–24, Nov. 2013.

[63] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson, "SynSin: End-to-end view synthesis from a single image," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7467–7477.

[64] G. M. Komaki and V. Kayvanfar, "Grey wolf optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time," *J. Comput. Sci.*, vol. 8, pp. 109–120, May 2015.

[65] A. Lachmann, D. J. B. Clarke, D. Torre, Z. Xie, and A. Ma'ayan, "Interoperable RNA-seq analysis in the cloud," *Biochim. Biophys. Acta, Gene Regulatory Mech.*, vol. 1863, no. 6, Jun. 2020, Art. no. 194521.

[66] P. Banerjee and S. Roy, "An investigation of various task allocating mechanism in cloud," in *Proc. 5th Int. Conf. Inf. Syst. Comput. Netw. (ISCON)*, Oct. 2021, pp. 1–6.

[67] Y. Yang and H. Shen, "Deep reinforcement learning enhanced greedy optimization for online scheduling of batched tasks in cloud HPC systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 3003–3014, Nov. 2021.

[68] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment," *J. Grid Comput.*, vol. 14, no. 1, pp. 55–74, Mar. 2016.

[69] S. Basu, M. Karuppiah, K. Selvakumar, K.-C. Li, S. K. H. Islam, M. M. Hassan, and M. Z. A. Bhuiyan, "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment," *Future Gener. Comput. Syst.*, vol. 88, pp. 254–261, Nov. 2018.

[70] R. Kobayashi and K. Adachi, "Radio and computing resource allocation for minimizing total processing completion time in mobile edge computing," *IEEE Access*, vol. 7, pp. 141119–141132, 2019.

[71] S. S. Murad, R. Badeel, N. Salih, A. Alsandi, R. Faraj, A. R. Ahmed, A. Muhammed, M. Derahman, and N. Alsandi, "Optimized Min-Min task scheduling algorithm for scientific workflows in a cloud environment," *J. Theor. Appl. Inf. Technol.*, vol. 100, no. 2, pp. 480–506, 2022.

[72] K. Z. Gao, Z. M. He, Y. Huang, P. Y. Duan, and P. N. Suganthan, "A survey on meta-heuristics for solving disassembly line balancing, planning and scheduling problems in remanufacturing," *Swarm Evol. Comput.*, vol. 57, Sep. 2020, Art. no. 100719.

[73] Y. Alayev, F. Chen, Y. Hou, M. P. Johnson, A. Bar-Noy, T. F. La Porta, and K. K. Leung, "Throughput maximization in mobile WSN scheduling with power control and rate selection," *IEEE Trans. Wireless Commun.*, vol. 13, no. 7, pp. 4066–4079, Jul. 2014.

[74] S. K. Panda, S. S. Nanda, and S. K. Bhoi, "A pair-based task scheduling algorithm for cloud computing environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 1, pp. 1434–1445, 2022.

[75] A. Sinha, P. Mishra, M. Ramish, H. R. Mahmood, and K. K. Upadhyay, "Employing unsupervised learning algorithm for stock market analysis and prediction," in *Proc. 1st Int. Conf. Adv. Comput. Future Commun. Technol. (ICACFCT)*, Meerut, India, Dec. 2021, pp. 75–79, doi: 10.1109/ICACFCT53978.2021.9837372.

[76] M. Ramish, A. Sinha, J. Desai, A. Raj, Y. S. Rajawat, and P. Punia, "IT attack detection and classification using users event log feature and behavior analytics through Fourier EEG signal," in *Proc. IEEE 11th Int. Conf. Commun. Syst. Netw. Technol. (CSNT)*, Indore, India, Apr. 2022, pp. 577–582, doi: 10.1109/CSNT54456.2022.9787637.

[77] A. Sinha, M. Ramish, S. Kumari, P. Jha, and M. K. Tiwari, "ANN-ANT-LION-MLP ensemble transfer learning based classifier for detection and classification of oral disease severity," in *Proc. 12th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Noida, India, Jan. 2022, pp. 530–535, doi: 10.1109/Confluence52989.2022.9734176.

[78] A. Sinha, B. Kumar, P. Banerjee, and M. Ramish, "HSCAD: Heart sound classification for accurate diagnosis using machine learning and MATLAB," in *Proc. Int. Conf. Comput. Perform. Eval. (ComPE)*, Shillong, India, Dec. 2021, pp. 115–120, doi: 10.1109/ComPE53109.2021.9752199.

[79] A. Raj, S. Jadon, H. Kulshrestha, V. Rai, M. Arvindhan, and A. Sinha, "Cloud infrastructure fault monitoring and prediction system using LSTM based predictive maintenance," in *Proc. 10th Int. Conf. Rel., INFOCOM Technol. Optim. (Trends Future Directions) (ICRITO)*, Noida, India, Oct. 2022, pp. 1–6, doi: 10.1109/ICRITO56286.2022.9964554.

[80] M. Bhargavi, A. Sinha, J. Desai, N. Garg, Y. Bhatnagar, and P. Mishra, "Comparative study of consumer purchasing and decision pattern analysis using pincer search based data mining method," in *Proc. 13th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Kharagpur, India, Oct. 2022, pp. 1–7, doi: 10.1109/ICCCNT54827.2022.9984410.

[81] M. Bhargavi, A. Sinha, G. M. Rao, Y. Bhatnagar, S. Kumar, and S. R. Pawar, "Application of IoT for proximity analysis and alert generation for maintaining social distancing," in *Key Digital Trends Shaping the Future of Information and Management Science* (Lecture Notes in Networks and Systems), vol. 671. Cham, Switzerland: Springer, 2023, doi: 10.1007/978-3-031-31153-6_2.

[82] B. Kumar, S. Roy, K. U. Singh, S. K. Pandey, A. Kumar, A. Sinha, S. Shukla, M. A. Shah, and A. Rasool, "A static machine learning based evaluation method for usability and security analysis in e-commerce website," *IEEE Access*, vol. 11, pp. 40488–40510, 2023, doi: 10.1109/ACCESS.2023.3247003.

[83] T. Hai, J. Zhou, Y. Lu, D. N. Jawawi, A. Sinha, Y. Bhatnagar, and N. Anumbe, "Posterior probability and collaborative filtering based heterogeneous recommendations model for user/item application in use case of IoVT," *Comput. Electr. Eng.*, vol. 105, Jan. 2023, Art. no. 108532, doi: 10.1016/j.compeleceng.2022.108532.

PALLAB BANERJEE was born in Ranchi, Jharkhand, India, in 1984. He received the B.Tech. degree in computer science and engineering and the M.Tech. degree in CSE from the Biju Patnaik University of Technology (BPUT), in 2008,and 2015, respectively. In 2008, he started his professional career as a Lecturer with the CSE Department, CIT, Ranchi. He is an Assistant Professor with the Department of Computer Science and Engineering, Amity University, Ranchi. More specifically, his work examines in the field of operating systems, real time operating systems, and cloud computing. He is currently an Assistant Professor with the Computer Science and Engineering Department, Amity University. He excelled in teaching field since last 11 years. His current research interests include task scheduling in cloud computing, RTOS, and algorithm design. He is a Life Member of the Computer Society of India.

SHARMISTHA ROY is currently an Associate Professor with the Faculty of Computing and Information Technology, Usha Martin University, Ranchi. Her research interests include nanotechnology, machine learning, robotics, software requirements ambiguity detection, system design, big data stream processing, and biomedical deep learning.

ANURAG SINHA received the bachelor's degree from Amity University, Jharkhand. He is currently pursuing the master's degree in computer science. He has published many research papers in IEEE conference, Springer, and Iop, also written chapter and research papers for Sci and Scopus journal. He was the finalist in Smart India Hackathon 2022; he has been shortlisted for his research based on the enhancement of medical imaging using deep learning. He has worked in industrial projects, patents, and Sci journal paper in field of nanotechnology, machine learning, robotics, software requirements ambiguity detection, system design, cloud based data warehousing, big data stream processing, natural language processing, wireless communications anomaly detection, and biomedical deep learning. He has won the Best Paper Award in research writing competition held at IIT Delhi. He was awarded The Young Scientist Award for research presentation. He is currently a Reviewer of *Journal of Applied Artificial Intelligence*. He has also been a Reviewer of IEEE conference Gucon 2020 and Springer conference at Northeastern Hill University. He is a regular reviewer of Sci and Scopus indexed conference journal *Applied Artificial Intelligence* (Taylor and Francis), Wireless Communications and Mobile Computing (Hindawi), MIS journal (Hindawi), and IEEE Guconnet conference series.

MD. MEHEDI HASSAN (Member, IEEE) received the B.Sc. degree in computer science and engineering from North Western University, Khulna, Bangladesh, in 2022, where he excelled in his studies and demonstrated a strong aptitude for research. He is currently pursuing the M.Sc. degree in computer science and engineering with Khulna University, Khulna. As the Founder and the CEO of the Virtual BD IT Firm and the VRD Research Laboratory, Bangladesh, he has established himself as a highly respected leader in the fields of biomedical engineering, data science, and expert systems. He is a dedicated and accomplished Researcher. His research interests are broad and include important human diseases, such as oncology, cancer, and hepatitis, as well as human behavior analysis and mental health. He is highly skilled in association rule mining, predictive analysis, machine learning, and data analysis, with a particular focus on the biomedical sciences. As a young researcher, he has published 33 articles in various international top journals and conferences, which is a remarkable achievement. His work has been well-received by the research community and has significantly contributed to the advancement of knowledge in his field. Overall, he is a highly motivated and skilled researcher with a strong commitment to improving human health and well-being through cutting-edge scientific research. His accomplishments to date are impressive, and his potential for future contributions to his field is very promising. In addition, he serves as a reviewer for 22 prestigious journals. He has filed more than three patents out of which two are granted to his name.

SHRIKANT BURJE received the M.E. degree from the Shri Sant Gajanan Maharaj College of Engineering (SSGMCE), Shegaon, Maharashtra, India, and the Ph.D. degree from CSVTU, Bhilai, Chhattisgarh, India. He is an Associate Professor with the Department of Electronics and Telecommunication Engineering, Christian College of Engineering and Technology (CCET), Bhilai. He has been engaged in research and teaching for more than 22 years. He has presented more than 20 articles in national and international journals and has two patents. His main area of interest includes microprocessor, microcontroller, embedded systems, circuit theory, basic electronics, and biomedical image processing.

ANUPAM AGRAWAL received the B.E. degree in electrical engineering from Rajasthan University, in 2009, and the M.Tech. degree in power systems from the Government College Bikaner, in 2014. He is currently pursuing the Ph.D. degree in electrical engineering from Manipal University Jaipur. He has more than nine years' experience in the teaching and research field, currently working as an Assistant Professor with the Electrical and Electronics Department, Bhilai Institute of Technology, Durg, Chhattisgarh, India. His research areas include solar energy, synthesis of materials for solar cells, fabrication and characterization of third-generation solar cell, and reliability and nodal pricing in power systems. He was a Project Fellow in the INDO-Russia Project (sanctioned by DST, Government of India, New India), in 2020. He has published 17 research articles in reputed SCI indexed journals (IF >5) and five in Scopus indexed peer-reviewed conferences. He also published two book chapters in Taylor & Francis Group and CRC Press. He also filed and published two utility patents and accepted three design applications in Intellectual Property India, Government of India.

ANUPAM KUMAR BAIRAGI (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science and engineering from Khulna University (KU), Bangladesh, and the Ph.D. degree in computer engineering from Kyung Hee University, South Korea. He is a Professor with the Computer Science and Engineering Discipline, KU. His research interests include wireless resource management in 5G and beyond, healthcare, the IIoT, cooperative communication, and game theory. He has authored and coauthored around 60 publications, including refereed IEEE/ACM journals and conference papers. He has served as a technical program committee member in different international conferences.

SAMAH ALSHATHRI received the bachelor's degree in computer science and the master's degree in computer engineering from King Saud University, Riyadh, Saudi Arabia, and the Ph.D. degree from the Department of Computer and Mathematics, Plymouth University, Plymouth, U.K. She is currently an Assistant Professor with the Department of Information Technology, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University (PNU), Riyadh. Her research interests include wireless networks, cloud computing, fog computing, the IoT, data mining, machine learning, text analytics, image classification, and deep learning. She was the Chair of the Network and Communication Department and participated in organizing many international conferences. She has authored or coauthored many articles published in well-known journals in the research field.

WALID EL-SHAFAI (Senior Member, IEEE) was born in Alexandria, Egypt. He received the B.Sc. degree (Hons.) in electronics and electrical communication engineering from the Faculty of Electronic Engineering (FEE), Menoufia University, Menouf, Egypt, in 2008, the M.Sc. degree from the Egypt-Japan University of Science and Technology (E-JUST), in 2012, and the Ph.D. degree from the FEE, Menoufia University, in 2019. Since January 2021, he has been a Postdoctoral Research Fellow with the Security Engineering Laboratory (SEL), Prince Sultan University (PSU), Riyadh, Saudi Arabia. He is currently a Lecturer and an Assistant Professor with the Department of Electronics and Communication Engineering (ECE), FEE, Menoufia University. His research interests include wireless mobile and multimedia communications systems, image and video signal processing, efficient 2D video/3D multi-view video coding, multi-view video plus depth coding, 3D multi-view video coding and transmission, quality of service and experience, digital communication techniques, cognitive radio networks, adaptive filters design, 3D video watermarking, steganography, encryption, error resilience and concealment algorithms for H.264/AVC, H.264/MVC, and H.265/HEVC video codecs standards, cognitive cryptography, medical image processing, speech processing, security algorithms, software-defined networks, the Internet of Things, medical diagnoses applications, FPGA implementations for signal processing algorithms and communication systems, cancellable biometrics and pattern recognition, image and video magnification, artificial intelligence for signal processing algorithms and communication systems, modulation identification and classification, image and video super-resolution and denoising, cybersecurity applications, malware and ransomware detection and analysis, deep learning in signal processing, and communication systems applications. He also serves as a reviewer for several international journals.