IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 December 2024, accepted 6 January 2025, date of publication 24 January 2025, date of current version 29 January 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3534031

RESEARCH ARTICLE

Agent-Based Adaptive Dynamic Round Robin (AADRR) Scheduling Algorithm

ZAFAR IQBAL KHAN[1], **MUZAFAR KHAN**[1], **AND SYED NASIR MEHMOOD SHAH**[2]
[1]Department of Software Engineering, National University of Modern Languages, Islamabad 44000, Pakistan
[2]Department of Computer Science, Institute of Space Technology, KICSIT Kahuta Campus, Islamabad 44000, Pakistan

Corresponding author: Zafar Iqbal Khan (zafar.khan@ist.edu.pk)

ABSTRACT Scheduling techniques are essential to increase resource utilization and task execution within modern computing environments. Round Robin Scheduling (RR) ensures a fair distribution of processes needing attention but often leads to inefficiencies in systems with heterogeneous tasks or different priorities due to large latency or resource usage differences. To address such problems, this paper introduces the Agent-based Adaptive Dynamic Round Robin (AADRR) process scheduling technique, which enhances process scheduling by continuously adjusting the time quantum and criteria, combining CPU burst time and priority. In the proposed AADRR, all processes are ranked dynamically by a software agent based on user preferences and current system load. This agent operates independently by keeping an eye on system parameters and making the required adjustments in real-time without requiring human intervention. We place processes in the queue according to their order of importance. A dynamic time quantum policy is suitable whenever it meets the mean duration of each process in the queue. Every round has the time quantum adjusted based on this method average burst time. AADRR highlights that the short processes are managed properly and the long processes are completed within a few rounds to fairly complete and maintain all the processes in the system. The proposed AADRR is more suitable for periodic tasks that employ a dynamic scheduling system and adapt time quantum according to the system state and job features. Additionally, AADRR efficiently manages preemptable tasks, using dynamic scheduling policies to accommodate variations in process priority and CPU burst times, ensuring fair scheduling, efficient resource utilization, and dynamic adaptability. To validate the effectiveness of the AADRR algorithm, we performed a comparative performance analysis with twelve other algorithms, including five traditional CPU scheduling algorithms and seven advanced job scheduling techniques, demonstrating optimal performance results. In our experiments, synthetic workload traces were generated using the Monte Carlo probability distribution method, which is scientifically recommended for creating diverse workload traces. Small, medium, and large datasets were used, with the small workload traces obtained from published studies and the large traces produced by the Monte Carlo simulation. AADRR efficiently reduces average turnaround times and average waiting times for each workload and performs better in response time. AADRR may not always provide the most favorable measures in all scenarios. Still, it performs better than other scheduling techniques in system performance, while being more efficient and flexible for different workloads.

INDEX TERMS CPU scheduling algorithms, round robin (RR), context switching (NCS), dynamic time quantum (DTQ), Monte Carlo probability distribution.

I. INTRODUCTION

System software, commonly referred to as an ‘‘operating system’’ (OS), helps application software and controls computer hardware. Operating systems are categorized into desktop OS, cluster OS, grid OS, cloud OS, etc. The

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

system serves as a midpoint between the user and the hardware by taking up the role of interacting with the system. Managing user tasks effectively is an operating system's primary objective. Operating systems have developed over time to enable multitasking, which has greatly increased performance from managing a single task at a time. A key component of improving and ensuring more dependable system results is CPU scheduling [1].

CPU scheduling is crucial in multicore computing systems' kernel modules, where a ready queue manages multiple processes. In two-process systems, each processor has a separate queue, and the operating system assigns tasks accordingly [2]. The digital transformation and smart computing era require efficient multi-user operations and multiprogramming, requiring quick and effective process completion and CPU scheduling measures for optimal performance [3].

Operating system process management involves dividing processor time into active tasks through RR scheduling, which is considered fair and simple [4]. Conventional techniques have a fixed time quantum, leading to excessive context shifts and delays in response times [5]. The RR is regularly preferred as the default scheduling method within real-time and multitasking operating systems because it is fair and can best provide a specified time slice or a time quantum for every function. However, inefficiencies could arise if the time quantum is not selected appropriately. Extremely small-time quantum can harm system performance since they raise average waiting and turnaround times [6].

Moreover, a constant-time slice is unable to adapt effectively when processes exhibit distinct burst times. As a result, processes could become slow or take longer than necessary to finish. To overcome the drawback of using a fixed-time quantum, which normally results in programs running longer than needed, the time quantum is varied dynamically [7]. The scheduling algorithm presents a dynamic time quantum for every cycle, where the processes are arranged by the computed rank order. This algorithm ensures that all activities are allowed for the dynamic time quantum period, thus ensuring that system responsiveness is enhanced and overheads associated with context switching are reduced [8]. The Agent-based Adaptive Dynamic Round Robin (AADRR) algorithm minimizes process TAT and improves CPU performance compared to the traditional RR algorithm. It performs better when it combines process priority with the adaptable time quantum.

II. RELATED WORK

The authors of [9] explored various CPU scheduling techniques, specifically Round Robin, FCFS, and SJF, intending to improve CPU utilization and performance parameters such as response, context switch, and turnaround times. Additionally, the Fixed-Time-Quantum strategies aimed at the reduction of large turnaround times are in place. The Self-Adjustment RR (SARR), a novel CPU scheduling technique, was developed by [10]. It is possible to adjust the current scheduled operating process burst time to the time quantum. This method addresses the issues of waiting and turnaround times by reducing context changes and turnaround times using a time quantum of ordinary round-robin procedure.

In [11], researchers studied the scheduling problem on a dedicated uni-processor system for a periodic set of jobs. For non-preemptive scheduling, he recommends using rate monotonic (RM) prioritization when jobs have deadlines. The ratio of task duration is inspected to determine scalability limits to ensure that each task is completed within its limits.

According to [12], a temporal RR scheduling method with a variable length time quantum. According to this method, throughput is said to be improved and NCS and AWT are minimized. However, EDRR doesn't consider task rank when adjusting time quanta, which reduces its effectiveness in schedules with different levels of estimation, particularly for completion time. Additionally, this technique troubles with heterogeneous workloads that include both short and long jobs since EDRR cannot handle jobs of different time and importance. As a result, it might result in more overhead, more instances of job failure and a lack of resources utilizations, especially when workloads change greatly in burst durations. In the case of the starvation problem present in the SJF algorithms, it is improved by a better CPU scheduling technique referred to [13]. It sorts according to the length of the burst, which helps improve AWT and ATAT.

Improving RR with Dynamic Time Quantum (IRRDQ) is a unique scheduling technique developed by [14]. In IRRDQ, using burst time and SJF techniques for optimizing the dynamic time quantum has led to a decrease in ATAT, AWT, and NCS. The study in [15] utilizes a new improved RR method (the CBRR) for decreasing CPU service time, including in particular AWT and ATAT. It considers processes with the same nature and for the number of processes in the group allocates time cracks.

In [16], the authors provide the construction of the Self-Adjustment RR (SARR), which can be defined as a new way of CPU scheduling. Currently, scheduled burst times for active processes appear to match the implemented time quantum. This method uses a simple round-robin approach with time quantum to address lead time and latency issues. SAAR helps minimize the NCS and reduces processing time.

According to [17], an algorithm named the Priority Based RR (PBRR) was presented. This algorithm aims to further improve the classical techniques of RR scheduling to cope with some drawbacks. The order in which processes are assigned to one another in the ready queue is also determined by priority as per the PBRR mechanism. Every process has an allocated priority index. This strategy performs better in AWT and ATAT than in basic RR.

The researchers in [18] present a method of CPU scheduling which is called Modified Priority Preemptive Scheduling (MPPST) which happens to be a new technique of CPU scheduling guarantees. In MPPST, levels of importance are due to the rotational turnover. This will also try to address the new findings of such a new strategy. Contrary to the conventional preemptive strategy, process scheduling was improved in addition to the problem of processes consuming the CPU. The two greatest scheduling methods, priority-based and RR scheduling, were explored by [19]. The authors present a novel technique called Mix Priority RR (PI-RR). While ensuring that both fairness and priority are adequately met, PI-RR achieves better waiting time, turnaround time, and starvation.

The study in [20] introduces an innovative CPU scheduling algorithm called Intelligent RR (IRR), which is the particular case of the proposed CPU scheduling portion, hence extending it. Intelligent RR partitions the processing queue into several sub-queues based on the similarity level of the C or T. This technique improves round-robin scheduling without utilizing queue depletion through time management. Since the IRR algorithm regulates the waiting time, processing time, and time-consuming context switches.

The Median-Average RR (MARR) is permitted in the study in [21]. Furthermore, the aim of the elastic time quantum is to enhance the average process's burst time. The MARR algorithm uses this technique to improve performance. Therefore, each implementation of MARR can be more than just a dynamic combination. Some static conversion methods change the amount of time to reduce waiting time (WT) and turnaround time (TAT).

In [22], the authors introduce a new approach called Dynamic Time Quantum Adjustment (DTQRR) and reduce RR scheduling based on a software-implemented CPU algorithm. As confirmed by simulations, DTQRR minimizes context switching and latency.

The study [23] presents a new method for dynamic quantum RR, the Differential Arrival Dynamic Quantum RR (DADQRR) algorithm. The objective is to reduce CPU scheduling for ATAT and AWT. The algorithm contains a time slice that is altered dynamically, which depends on a few factors. However, in one case, ATAT performed slightly better than AWT concerning context switching.

According to [24], a new hybrid scheduling algorithm consisting of RR and SJF scheduling is presented. Low and high-complexity round-robin strategies are considered. It is shown that the performance of the modified algorithm improves when tasks are completed before their assigned quantum expires, instead of temporarily pausing and then resuming them.

According to [25], an improved RR scheme for real-time operating systems (RTOS) serves as the basis for a new RR-type scheduling algorithm. This algorithm alleviates the shortcomings of orthodox RR approaches, enhancing system performance. It focuses on the reduction of context-switching periods and response times.

In [26], the authors compare and analyze the standard RR techniques with enhanced RR techniques. It has been found that the enhanced algorithm improves effectiveness because more tasks are being finished in shorter sprints than the quantum allows before the process is ordered to be returned to the queue. Applying this technique increases system throughput at the expense of AWT, ATAT, and NCS being reduced.

The Study in [27] proposed the utilization of the Harmonic Arithmetic Mean (HARM) technique in data mining, particularly in defining the segment of time when the round-robin scheduling activities happen. It reduced the WT and TAT increased the efficiency of the Time Quantum. Moreover, [28] claims that the performance of RR scheduling can further be improved if it's used in combination with SJF scheduling. The authors suggest dividing employment by time duration and dividing the sum evenly by the median.

Reference [29] suggested the use of a DTQ for the RR algorithm, which improves CPU scheduling by taking process computing burst times into account. However, [30] discusses the RR Dynamic Time Quantum (RRDTQ) modifies the time quantum by executing each RR over the average process burst time. By using this method, the amount of AWT and ATAT between context switches and their derivatives is decreased.

The authors in [31], proposed a Modified Maximum Urgency First (MMUF) to address dynamic real-time task systems' critical task losses using schedulability mechanisms. This further increases the efficiency and reliability of the system through a decrease in the number of task preemptions and the required amount of rescues of the tasks, especially when the CPU is highly loaded. Unlike approaches that deploy agents for decision-making, MMUF prioritizes task urgency but lacks certain dynamic adjustment capabilities.

Recent trends in development of scheduling algorithms aim to achieve flexibility as well as efficiency in the operations particularly in dynamic and heterogeneous computing environments. Despite the fact that this condition is prevalent in a number of the existing approaches, it has been difficult to carry out the equity maintenance while adjusting to varying workloads requirements. These problems are at least in part solved by the AADRR algorithm which uses agents to make the decisions and alters time constraints, thus offering also a great possible of application in more complex systems. Table 1 gives a summary of various algorithms and how well they work in terms of improving CPU scheduling efficiency.

III. PROPOSED SCHEDULING ALGORITHM

We propose an algorithm for scheduling that we call Agent-based Adaptive Dynamic Round Robin (AADRR). The AADRR enhances the traditional RR with several new features: dynamic time quantum (DTQ), a ranking mechanism for processes, adaptability to mixed workloads, and agent-based decision-making. The time quantum for every task is adjusted automatically founded upon factors like task importance and how long the CPU needs to work on it. This helps the system handle different types of tasks better. The feature of dynamic time quantum adjustment of the AADRR algorithm allows it to respond to different levels of processing demand, which is even more relevant in the case

TABLE 1. Comparison of scheduling techniques and performance metrics.

Ref	Proposed Technique	Performance Metrics	Findings
[10]	Self-Adjustment Time Quantum (SAAR)	WT, TAT	Improves performance compared to static time quantum.
[11]	Priority Non-Preemptive (P-NP)	Utilization bound, Schedulability	RM works best for non-preemptive scheduling.
[12]	Enhanced Dynamic RR (EDRR)	NCS, AWT, ATAT	It reduces WT, TAT and NCS.
[13]	Improved Scheduling Strategy	AWT, ATAT	Reduces starvation and improves the WT for extended burst times.
[14]	Improving RR with Dynamic Time Quantum (IRRDQ)	AWT, ATAT, NCS	Performance is improved with IRRDQ over static time quantum.
[15]	Clustering-Based RR Algorithm (CBRR)	AWT, ATAT, NCS	Clustering improves scheduling efficiency and reduces AWT.
[16]	SJFDRR (Smart Job First Dynamic RR)	AWT, ATAT, NCS	It offers improved performance over traditional methods.
[17]	Priority Based RR (PBRR)	AWT, ATAT	Decreases AWT and ATT in comparison to traditional RR
[18]	Modified Priority Preemptive Algorithm (MPP)	Mean TAT, WT	Dynamic priority adjustments improve scheduling performance.
[19]	Mix Priority RR (PI-RR)	Efficiency, Fairness	Combines benefits of both algorithms to address starvation
[20]	New Intelligent RR (NIRR)	RT, Throughput	Intelligent adjustments improve scheduling performance.
[21]	Median-Average RR (MARR)	TAT, WT	Median-average adjustments improve efficiency.
[22]	Dynamic Time Quantum Adjustment (DTQRR)	Scheduling Performance, WT	Based on workload improve performance.
[23]	Different Arrival-Dynamic Quantum RR (DADQRR)	AWT, ATT, RT, NCS	Reduces the quantity of RQ (Ready Queue) procedures.
[24]	A hybrid of RR and SJF	AWT	It reduces waiting time compared to traditional methods.
[25]	Modified RR with Priority-Based Intelligent Time Slicing	WT, RT, Throughput	Time slicing, and priority-based scheduling are combined to improve the traditional RR.
[26]	Intelligent Time Slice	Time slice efficiency, RT	Improve time slice for better job management.
[27]	HARM (Harmonic-Arithmetic Mean)	RT, ATAT	Enhanced performance in RR scheduling.
[28]	SJF and RR Hybrid with dynamic quantum	ART, task throughput	Increased efficiency and task handling.
[29]	Amended Dynamic RR	Response time, system throughput	Improved task scheduling and reduced AWT.
[30]	RR with Dynamic Time Quantum (RRDTQ)	ATAT, AWT, NCS	Utilizes a DTQ.
[31]	Modified Maximum Urgency First (MMUF)	Task Preemption Rate, Overhead, Task Failure Rate, Deadline Misses	Reduces task preemption and overhead, performs better when the CPU load factor,leads to fewer non-critical tasks failing in overloaded conditions.

of distributed systems that exhibit high workload variability. This approach ensures fair resource allocation, which is a necessity for clouds and multi-tenant systems where fairness among tasks and users is a important concern.

Tasks are sorted by their importance, so important tasks get more CPU time and are managed more effectively. AADRR uses agents to make decisions about task scheduling on their own. These agents can change how tasks are scheduled based on current system conditions, without needing human help. AADRR can handle a mix of short and long tasks more efficiently by changing its scheduling settings based on the task's specific traits.

AADRR first computes the rankings based on the given criteria and then sorts the processes based on these rankings. Sorting is carried out once in each scheduling cycle, and processes are then executed for a given value of time quantum or dynamic time quantum. Task priority or system load is only sorted when there is a need to change the process ranking, and our method allows sorting only when necessary. The best sorting algorithm, Quick Sort, has been implemented, as it is not a time-consuming algorithm. The system does not generate too much overhead, and this sorting process has been accounted for in terms of its overhead, with the frequency and time impact explained further in the manuscript. Sorting is only carried out in each cycle when new jobs arrive in the system.

Proposed Algorithm Procedure:

The proposed algorithm includes the following steps:

1. **Initialization:**
   Place all processes in a circular ready queue.
2. **Rank Calculation:**
   In the ready queue, find the rank of each process, and apply the formula:

   $$\text{Rank} = A \times \text{CPU burst time} + B \times \text{priority number}$$

   The values of A and B could be user inputs or environmental variables that prioritize CPU-bound jobs, priority jobs, or a mixed nature.
3. **Sorting:**
   The procedures in the ready queues are sorted in ascending order using their rank values as a guide.
4. **Compute Dynamic Time Quantum:**
   a. Find out how long each process CPU burst will take to reach the ready queue.
   b. The median CPU burst times should be found. Applying this median value yields the dynamic time quantum for the current cycle.
5. **Process Execution:**
   a. Choose from the sorted ready queue the process with the lowest rank.
   b. For the length of the calculated dynamic time quantum, dedicate the CPU to this task.
6. **Completion Condition:**
   a. Permit the shortened process to conclude if the square root of the dynamic time quantum is less than or equal to the CPU burst duration that remains.
   b. Prioritize the process, and after the dynamic time quantum finishes, move it to the end of the sorted queue.
7. **Recalculation:**
   a. Before starting the next cycle, recalculate the process rank in the queue.
   b. With the process median CPU burst time, the dynamic time quantum was recalculated.
8. **Repetition:** Continue doing this until all of the processes have been performed.

Figure 1, presents the flowchart for the proposed AADRR scheduling algorithms. This flowchart offers a visual representation of the algorithm's logic and structure, showing how it analyzes data, makes decisions, and proceeds from one step to the next.

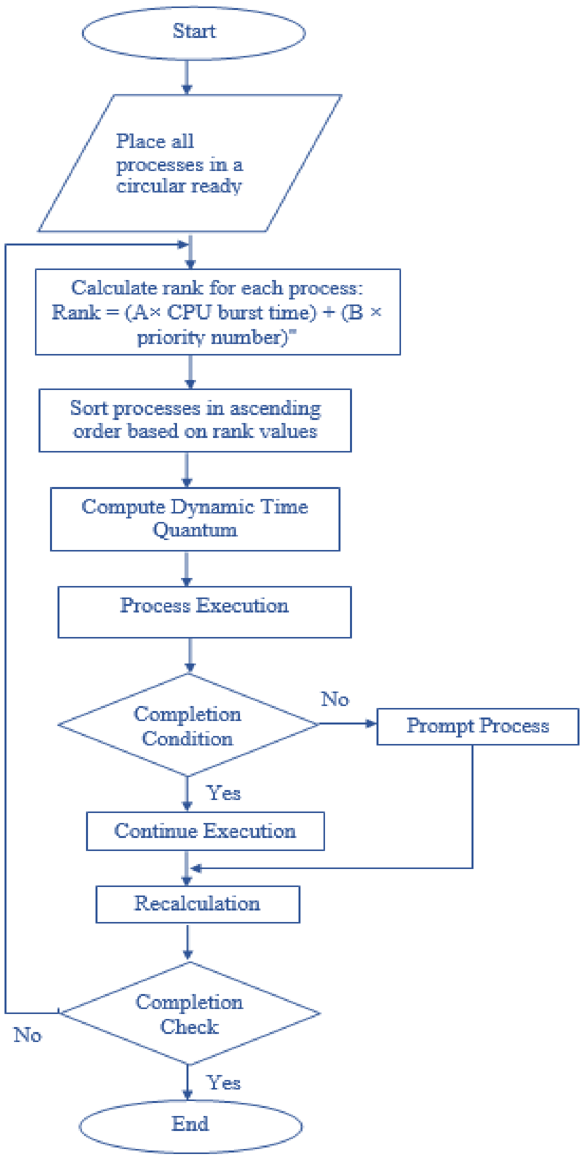
Start
Place all processes in a circular ready
Calculate rank for each process: Rank = (A× CPU burst time) + (B × priority number)"
Sort processes in ascending order based on rank values
Compute Dynamic Time Quantum
Process Execution
Completion Condition
No
Prompt Process
Yes
Continue Execution
Recalculation
Completion Check
No
Yes
End

FIGURE 1. Proposed scheduling algorithm flowchart.

These ranks are used to dynamically modify the scheduling criteria, increasing efficiency and performance all around. The proposed strategy starts by inserting all processes into a circular ready queue. Finally, each process is evaluated according to the following formula:

$$\text{Rank} = A \times \text{CPU burst time} + B \times \text{priority number}$$

where A and B are parameters previously set by the peripheral or the user himself and associated with a specific task. In this particular context, once the ranking is performed, the corresponding processes are placed in ascending order. The algorithm can determine the DTQ for each cycle by using the CPU burst times of these queued processes and calculating the median value. The DTQ however is employed in the section where the user describes how the resource performance tracking system operates during the DTQ period.

Resources are allocated to the particular process whose rank is the lowest during the DTQ period. The remaining CPU burst time must be completed once it is less than or equal to the square root of the DTQ. In other circumstances, normally when its quantum expires it is pushed to the front of the queue. In the next and every subsequent cycle, ranks as well as the DTQ are recalculated based on the median CPU burst. All these have to be done, and the last one is approaching the end of every step.

AADRR algorithm is a new process scheduling method, enhances round-robin scheduling to make it more adaptive, which integrates a ranking system with dynamic time slice allocation. AADRR assigns a task to a queue with a particular priority and estimates the CPU burst time, ensuring that the optimum scheduling of needs of the queuing system are met. AADRR is ideal for usage in the environments that contain preemptive and periodic tasks and has the ability to perform smooth transitions during shifts in workload. Continuous changes are made in the system in response to the accuracy of the execution which results in the preemption of small jobs being avoided and the scheduling of bigger ones to be fair. Moreover, the algorithm is adaptive enough to satisfy a demanding workload or tight deadlines employed to manage with periodic tasks.

A. ILLUSTRATIVE EXAMPLE

An example provided a clearer understanding of the proposed method. Let's discuss the proposed AADRR algorithm by taking a problem scenario. Let's consider an example with three processes as demonstrated in Table 2.

TABLE 2. Process scheduling (3 Processes).

Process Id	CPU Time	Priority No
P_1	8 ms	2
P_2	4 ms	1
P_3	6 ms	3

Let A = 10 and B-5

Step-by-Step Algorithm Execution:

1) **Rank Calculation:**
   - Rank of $P_1 = 10 \times 8 + 5 \times 2 = 90$
   - Rank of $P_2 = 10 \times 4 + 5 \times 1 = 45$
   - Rank of $P_3 = 10 \times 6 + 5 \times 3 = 75$
2) **Sorting:**
   - Sorted process order: $\{P_2, P_3, P_1\}$
3) **Compute Dynamic Time Quantum:**
   - CPU burst times: $[4, 6, 8]$
   - Median CPU burst time: 6 ms
   - Dynamic time quantum: 6 ms
4) **Process Execution:**
   - P_2 executes for 4 ms (remaining burst time $\leq \sqrt{6}$ ms, so it completes)
   - Recalculate ranks and sort the remaining processes: P_3 (Rank 75), P_1 (Rank 90)
   - Dynamic time quantum recalculated based on new median (since only P_3 and P_1 remain):
     - CPU burst times: $[6, 8]$
     - Median CPU burst time: 7 ms
     - Dynamic time quantum: 7 ms
5) **Continue Execution:**
   - P_3 executes for 6 ms (remaining burst time $\leq \sqrt{7}$ ms, so it completes)
   - Only P_1 remains: P_1 executes for 8 ms and completes

This example shows how to implement an AADRR scheduling algorithm, ensuring better process prioritizing and efficient CPU use by adapting to the changing workload and process priorities.

B. FEATURES OF PROPOSED SCHEDULING ALGORITHM (AADRR)

The following are the features of the proposed AADRR algorithm:

- **Balancing Fairness and Efficiency:** The proposed scheduling algorithm could reconcile the requirements for more critical processes to be prioritized (taking priority numbers into account) and fair CPU allocation (taking CPU burst times into account).
- **Improved Process Prioritization:** More critical Processes (higher priority) are given more weight, ensuring that important tasks are not delayed excessively. Values of 'B' in the rank could be taken as 'high' as per the needs for priority considerations. At the same time, processes with shorter burst times can be processed quickly, reducing waiting times and improving system responsiveness.
- **Adaptive Scheduling:** This combined metric allows the scheduling algorithm to adapt to varying workloads. Processes with short burst times and high priorities will naturally rise in rank, ensuring that they get CPU time sooner, which can be particularly useful in real-time and interactive systems.
- **Reduced Starvation:** The combined metric lowers process starvation which refers to the indefinite waiting of the low-priority processes by the other processes in the system. Since burst time is in consideration, the algorithm avails a situation where one averagely finishes the work. It is expected that different processes will be carried out depending on their work demands.
- **Enhanced Throughput and Turnaround Time:** This is because there is improved allocation of the processes by the order and the burst time and as a result the algorithm will enhance the overall throughput (number of processes completed in a specified period) and lower the average turnaround time (time between submission and finishing a task).

- **Context Information Executed:** Because of the continuing nature of the rank recalculation as the processes are added into the system or removed from the queue hence the scheduling is also modified this makes it more flexible to adapt to any real-time factor that may affect the scheduling.
- **Scalability:** This combined metric generally has the attribute of not being limited to a specific workload structure. Still, it can accommodate almost all forms and characteristics of biases that exist in processes, computing, or input/output- regarding their priority.

C. PERFORMANCE COMPARISON OF TRADITIONAL RR AND AADRR

Our proposed AADRR is compared with the traditional RR regarding WT, TAT, RT, and some important metrics. This theoretical analysis shows how AADRR outperforms traditional RR scheduling, particularly when managing varying process burst times and dynamically updating the time quantum.

1) PROCESS EXECUTION AND EXECUTION ORDER IN AADRR

Process execution in AADRR describes how the proposed algorithm carries out processes highlighting ranking and dynamic time quantum techniques to increase scheduling effectiveness. Table 3 presents the Process Execution and Execution Order in AADRR for each of the six processes (P1 through P6). It illustrates how each operation operates sequentially within its allotted time window as well as when it ends. The spreadsheet allows you to keep track of the process sequence, total time spent running each, and completion time for each.

2) PROCESS SCHEDULING METRICS USING AADRR

Table 4 shows key performance metrics such as WT, TAT, and RT of every process one by one to analyze the actual performance effect for the AADRR Scheduling Algorithm.

- **TAT:** The term TAT describes how long it takes to complete a process from receipt and is calculated using the formula:

$$\text{TAT} = \text{Completion Time} - \text{Arrival Time} \tag{1}$$

Since the arrival time is 0 for all processes:

$$\text{TAT} = \text{Completion Time} \tag{2}$$

- **WT:** WT shows how long a process needs to stay in the queue. This is the amount of time that passes before processor time is allotted, and is calculated using the formula:

$$\text{WT} = \text{TAT} - \text{CPU Burst Time} \tag{3}$$

- **RT:** RT is the duration between first receiving the application and first executing the application, since all processes arrive at time 0 and are scheduled based on their rank, the response time for each process can be tracked by the order in which they are executed, and is calculated using the formula:

$$\text{RT} = \text{Execution Start Time} - \text{Arrival Time} \tag{4}$$

Since the arrival time is 0 for all processes:

$$\text{RT} = \text{Execution Start Time} \tag{5}$$

The AADRR algorithm effectively ranks the processes and adjusts the time quantum dynamically with rank to achieve a better overall system performance. The ATAT for the six processes was 20.17 ms, with an AWT of 13.5 ms and ART of 13.5 ms, resulting in enhanced process management. The AADRR Scheduling Algorithm deals with scheduling and makes changes in terms of priority values to optimize performance for user-level processes.

3) PROCESS SCHEDULING METRICS USING TRADITIONAL RR

Table 5 gives an elaborate view of process scheduling using the Traditional RR algorithm, showing critical parameters for every process (P1 to P6). This summarizes how long it took to complete them overall. It provides the initial execution start time of each process, which represents when a process begins running for the first time during its scheduling cycle. The execution timeline is converted into several rounds, where the remaining burst time for each process is recalculated. The process's termination time is recorded in the completion time, which is the entire period since its submission. This indicates the overall time a process takes from start to end. WT is also available, calculated as the difference between TAT and CPU burst time, demonstrating a variation in the number of processes in the ready queue that are currently waiting before execution.

The summary metrics include AWT (26.17ms), ART (10 ms), and ATAT (32.83 ms). The metrics provide meaningful information about how well the Traditional RR algorithm manages and schedules processes, as well as where the enhancements can be employed for improving process execution and system performance. The modified Approach, incorporating dynamic time quantum and process ranking, is used in conjunction with a modified version of RR.

4) COMPARISON OF SCHEDULING TECHNIQUES

Round-robin scheduling is the simplest and fairest CPU scheduling algorithm. RR switches on at each process and the CPU takes one turn at each process by a fixed time. However, Round-robin scheduling has some restrictions in the execution of processes with varying burst times. To solve all restrictions of standard rounds, AADRR is a better scheduling method. This method sets specifications and also automatically adjusts time quantum according to processes' median burst time by varying estimated burst time and previous insert activity. Table 6 summarizes the advantages and disadvantages of performance measures of Scheduling Techniques.

TABLE 3. Process execution and execution order in AADRR.

Process	CPU Burst Time (ms)	Priority Number	Execution Start Time (ms)	Execution End Time (ms)	Completion Time (ms)
P1	10	2	30	40	40
P2	5	1	0	5	5
P3	8	3	22	30	30
P4	6	2	9	15	15
P5	4	4	5	9	9
P6	7	2	15	22	22

TABLE 4. Process scheduling metrics using AADRR.

Process	CPU Burst Time (ms)	Arrival Time (ms)	Execution Start Time (ms)	Execution End Time (ms)	Completion Time (ms)	TAT (ms)	WT (ms)	RT (ms)
P1	10	0	30	40	40	40 - 0 = 40	40 - 10 = 30	30 - 0 = 30
P2	5	0	0	5	5	5 - 0 = 5	5 - 5 = 0	0 - 0 = 0
P3	8	0	22	30	30	30 - 0 = 30	30 - 8 = 22	22 - 0 = 22
P4	6	0	9	15	15	15 - 0 = 15	15 - 6 = 9	9 - 0 = 9
P5	4	0	5	9	9	9 - 0 = 9	9 - 4 = 5	5 - 0 = 5
P6	7	0	15	22	22	22 - 0 = 22	22 - 7 = 15	15 - 0 = 15

Summary for AADRR Scheduling: (ATAT: 20.17 ms, AWT: 13.5 ms and ART: 13.5 ms)

TABLE 5. Process scheduling metrics using traditional RR.

Process	CPU Burst Time (ms)	First Execution Start Time (ms)	Total Execution Time (ms)	CT (ms)	TAT (ms)	WT (ms)	RT (ms)
P1	10	0	10	41	41 - 0 = 41	41 - 10 = 31	0 - 0 = 0
P2	5	4	5	29	29 - 0 = 29	29 - 5 = 24	4 - 0 = 4
P3	8	8	8	33	33 - 0 = 33	33 - 8 = 25	8 - 0 = 8
P4	6	12	6	35	35 - 0 = 35	35 - 6 = 29	12 - 0 = 12
P5	4	16	4	20	20 - 0 = 20	20 - 4 = 16	16 - 0 = 16
P6	7	20	7	39	39 - 0 = 39	39 - 7 = 32	20 - 0 = 20

Summary for traditional RR Scheduling : (ATAT: 32.83 ms, AWT: 26.17 ms and ART: 10 ms)

From The above discussion and analysis of results, Traditional RR Scheduling provides simplicity and fairness but may suffer from inefficiencies in scenarios with diverse process requirements and burst time. AADRR optimizes CPU utilization, reduces waiting time, and improves overall system performance by dynamically adjusting the time quantum and prioritizing processes based on a combined rank metric. The AADRR (A) and (B) values were selected due to the jobs' nature and the goal of achieving a balance between priority and CPU burst time.

The analysis of traditional RR scheduling in comparison to the proposed AADRR highlights numerous key differences and enhancements. Traditional RR uses a fixed time quantum and arranges processes in a circular schedule according to arrival times. Because it gives an equal quantity of CPU time to each task, it may not be effective for tasks with different burst times which raises ATAT and AWT. On the other hand, AADRR integrates a ranking system that takes into account both burst time and priority for scheduling and it applies a dynamic time quantum that modifies according to

TABLE 6. Comparison of scheduling techniques.

Aspect	Traditional Round-Robin Scheduling	Proposed Scheduling Algorithm (AADRR)
Time Quantum	Fixed (e.g., 4 ms)	Dynamic (based on median burst time)
Scheduling Order	Circular, based on arrival time	Based on rank (combined metric of burst time and priority)
Completion Condition	Fixed time quantum; no special completion rules	The process can be completed if the remaining burst $\leq \sqrt{\text{dynamic time quantum}}$
ATT	32.83 ms	20.17 ms
AWT	26.17 ms	13.5 ms
ART	10 ms	13.5 ms
Pros	• Fair to all processes • Simple and easy to implement • Guarantees the same amount of CPU time for every process	• Quantum of dynamic time based on median burst time • Better ART and AWT • Better handling of processes with varied burst times
Cons	• Fixed time quantum may be inefficient • High context switching overhead • Lead to high waiting time for longer burst processes	• More complex to implement • Overhead for calculating dynamic time quantum and ranking • Requires accurate ranking and burst time calculations

the process's median burst times. This method greatly reduces ATT and AWT while enabling more efficient management of processes with different burst times. Because dynamic time quantum and ranking must be calculated AADRR implementation is more complex but the result is improved performance metrics: ATAT (20. 17 ms) and AWT (13. 5 ms) are lower than with Traditional RR (ATAT: 32. 83 ms AWT: 26. 17 ms).

We performed several experiments using different values of A and B to compute the rank depending upon dynamic requirements from users or systems to give priority to shorter jobs, longer jobs, mixed nature, prioritized jobs, etc. We presented the findings in below mentioned Table 7; which will be later explained in detail in further experiments using real-time data sets.

5) PERFORMANCE AND EFFICIENCY PARAMETERS

Table 7 shows the combinations that allow the AADRR scheduling algorithm to be tailored to different types of workloads, enhancing performance and efficiency based on specific system requirements. From the below table, it is concluded that the values of A = 10 and B = 5 have shown good performance measures.

IV. EXPERIMENTAL PERFORMANCE ANALYSIS

In the "Experimental Performance Analysis", we compare the proposed AADRR algorithm against several scheduling techniques, including RR, P-NP, P-P, SJF, FCFS, SARR, NRR, NIRR, PBRR, MRR, DTQRR, and MARR. The analysis evaluates each scheduling technique against key performance metrics such as AWT, ATAT, and ART, highlighting the strengths and potential benefits of the proposed AADRR algorithm in optimizing scheduling efficiency and system performance.

AADRR uses agents to make decisions, ranks processes, adjusts time quantum dynamically and employs a weighted ranking formula that takes priority and CPU burst into account, frequently recalculating ranks. AADRR uses agents to automatically adjust task scheduling according to system conditions.

A. BASELINE APPROACHES

- Round Robin (RR) [6], [7], [9], [11], [16], [24], [28]
- Priority Non-Preemptive (P-NP) [11]
- First Come First Served (FCFS) [7], [9]
- Priority Preemptive (P-P) [18]
- Shortest Job First (SJF) [9], [13], [16], [24], [28]
- Self-Adjustment-RR (SARR) [10]
- New RR (NRR) [25]
- Priority Based RR (PBRR) [17]
- Dynamic Time Quantum RR (DTQRR) [22]
- New Intelligent RR (NIRR) [20]
- Modified RR (MRR) [23]
- Modified Adaptive RR (MARR) [21]

B. PROPOSED SCHEDULING ALGORITHM

This experiment presented the Agent-based Adaptive Dynamic Round Robin (AADRR) scheduling algorithm.

C. AVERAGE WAITING TIME

The performance parameters (i.e., AWT)of many process sets can vary significantly depending on the different scheduling algorithms used. Table 8 shows the AWT corresponding

TABLE 7. Performance and efficiency of AADRR with varying parameters.

Combination	AWT	ATAT	ART	Context Switching	Starvation	Suitability
(A=1, B=0)	High for longer jobs and low for shorter jobs	High for longer jobs and low for shorter jobs	High for longer jobs and low for shorter jobs	High for longer jobs	High for low-priority jobs	Short CPU burst time jobs
(A=0, B=1)	Give low priority to jobs that have a high priority. Next, return their priority to high.	Lower the rank given to high-priority jobs. Next, return their priority to high.	Lower the rank assigned to high-priority jobs. Next, return their priority to high.	Low for jobs with a high priority.	High for jobs with a low priority.	High-priority jobs
(A=10, B=5)	Balanced, moderate for all jobs	Balanced, moderate for all jobs	Balanced, moderate for all jobs	Moderate	Minimal	Mixed nature jobs
(A=5, B=5)	Moderate for all jobs	Moderate for all jobs	Moderate for all jobs	Moderate	Minimal	Mixed nature jobs
(A=7, B=3)	Lower for short jobs	Lower for short jobs	Lower for short jobs	Moderate to high	Moderate for low-priority jobs	Short CPU burst time jobs with some priority consideration
(A=3, B=7)	Lower the importance of high-priority tasks	Lower the importance of high-priority tasks	Lower the importance of high-priority tasks	Low to moderate	Moderate for short burst jobs	High-priority jobs with consideration for burst time
(A=8, B=2)	Lower for short jobs	Lower for short jobs	Lower for short jobs	Moderate	Moderate for low-priority jobs	Jobs with emphasis on reducing ATAT
(A=6, B=4)	Balanced	Balanced	Balanced	Moderate	Minimal	Balanced workload with moderate priority emphasis

to various scheduling algorithms, as well as the proposed AADRR algorithm for different process sets. In AADRR, the time slices or quantum of the processes is allocated in real time depending on the system state as well as the process characteristics. The term ‘agent-based’ means where agents or decision-making entities exist in the system that constantly change rules set across the schedules, including but not limited to the statuses of the processes or the load on the system.

For the smaller process sets (ranging from 1 to 6 processes), AADRR is successful all the time, which indicates the proper use of dynamic resource management. The algorithm’s performance varies more with larger process sets, possibly because managing dynamic time slices adds complexity and overhead. Small to moderately sized process sets benefit greatly from the flexible and adaptive scheduling of AADRR.

Larger process sets (ranging from 7 to 9 processes) typically have higher values for FCFS and RR, suggesting that these algorithms are less effective at managing complex processes compared to AADRR. While they may be more flexible than AADRR, SJF, and P-NP may only be optimized for specific situations. MRR, DTQRR, and MARR may exhibit different performance characteristics depending on their design. However, AADRR appears to manage a variety of conditions with ease.

Similarly, Fig. 2 shows a comparison of twelve (12) scheduling techniques and the proposed algorithm using AWT for small process sets and Fig. 3 shows a comparison of medium and large process sets. The AADRR values for Process Sets (1 to 6) are significantly lower compared to algorithms such as FCFS, RR, and others. This suggests that AADRR performs particularly well for smaller, simpler process sets. This indicates that even though AADRR adapts dynamically, its effectiveness may be impacted by the overhead or difficulty of managing such extensive operations. When comparing AADRR against many conventional and advanced scheduling algorithms, it consistently shows lower average waiting times across a variety of process sets. Overall, it is more efficient at controlling process waiting times than most algorithms.

1) PERFORMANCE ANALYSIS OF SCHEDULING ALGORITHMS BASED ON AWT

- **Best Performance:** SJF and AADRR consistently achieve the lowest average waiting times across all process sets, making them ideal for minimizing waiting times in both small and large process sets. SJF is particularly efficient due to its focus on the shortest jobs, which reduces overall waiting times, especially in smaller process sets.
- **Medium Performance:** FCFS, P-NP, and RR (Round Robin) provide moderate performance. These algorithms generally exhibit higher waiting times compared to SJF and AADRR, especially as the number of processes increases. RR’s time-sharing nature results in fair but higher waiting times due to frequent context switching, which becomes more pronounced in larger process sets.

TABLE 8. A comparison of twelve (12) scheduling techniques and the proposed algorithm using AWT.

Process Set	Proposed (AADRR)	RR	P-NP	P-P	SJF	FCFS	SARR	NRR	NIRR	PBRR	MRR	DTQRR	MARR
Process Set 1	48.5	147.250	100.00	100.000	48.5	100.00	112.5	110	142.5	92.5	61	48.5	48.5
Process Set 2	30.5	115.500	85.50	85.500	30.5	85.50	96	98.5	110.5	59.5	43.75	30.5	30.5
Process Set 3	35.75	115.500	81.000	81.000	35.75	81.00	81	90.25	114.75	71.75	47.25	35.75	35.75
Process Set 4	22.75	98.00	75.50	75.500	22.75	75.50	75.5	88.5	94.25	45.75	36	37.75	37.75
Process Set 5	14.8	44.00	43.600	43.600	26.2	43.20	43.2	45.2	38.4	30.4	30.6	28.6	28.6
Process Set 6	14.8	44.00	42.200	42.200	26.2	43.20	43.2	37.2	37	30.4	30.6	28.6	28.6
Process Set 7	40237.8	86741.2	128257.4	128218.7	41415.7	109385.1	84955.1	109384.1	87037.8	80395.9	88276.9	78817.9	86569.3
Process Set 8	200550.9	403075.3	639578.3	639516.3	201588.7	603692.7	393539.4	603692.7	404241.0	400825.8	491331.6	391105.6	441727.5
Process Set 9	407743.4	816508.5	1227935.6	1230291.3	406914.3	1252419.8	934702.9	1252419.8	816097.3	813671.6	988823.6	932214.5	958144.0

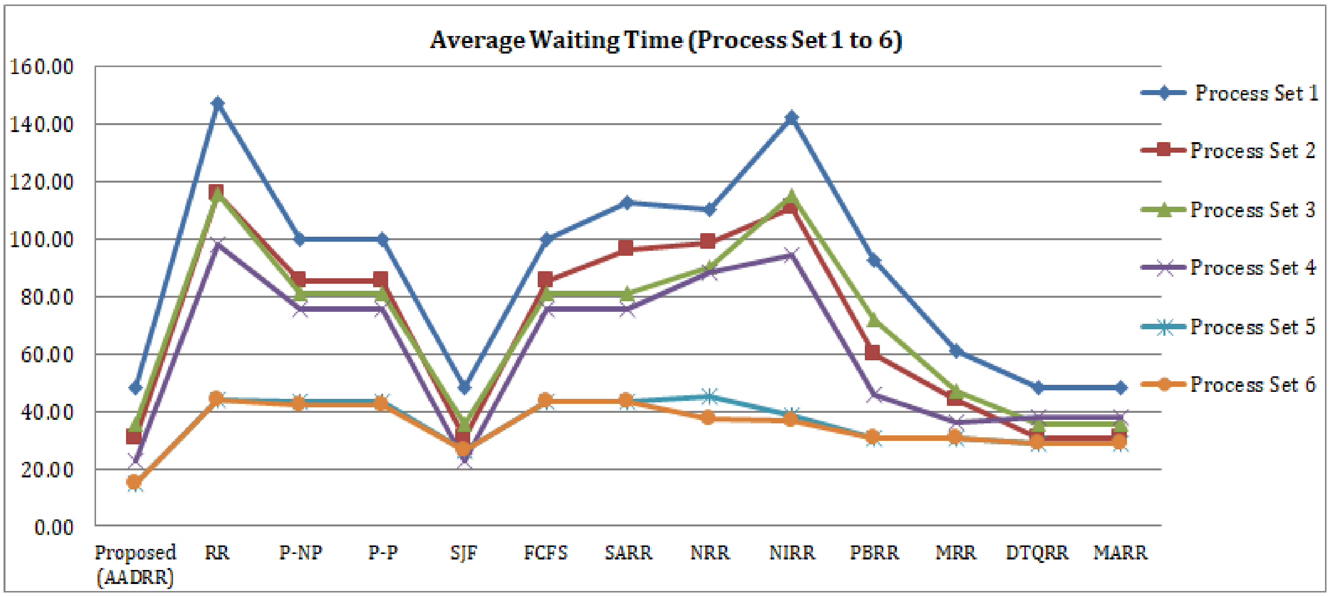

FIGURE 2. AWT for small process set.

- **Worst Performance:** P-P, NRR, and MRR exhibit poor performance, particularly in larger process sets (Sets 7, 8, and 9). These algorithms struggle with higher average waiting times due to inefficiencies in managing larger workloads and priorities.

AADRR shows consistently high performance across all process sets, especially when compared to traditional RR and FCFS. Its ability to adaptively manage job scheduling based on dynamic priorities and workloads makes it a robust choice, particularly in distributed and cloud environments. AADRR ensures a balance fairness and efficiency by automatically adjusting time intervals based on process priorities and workloads. AADRR guarantees that every task receives a fair amount of the CPU's time, reduces the AWT, and utilizes resources as efficiently as possible, especially for small and medium-sized process sets. For scenarios requiring scalable and efficient scheduling, AADRR is recommended due to its balanced approach between minimizing waiting time and enhancing process management efficiency.

D. AVERAGE TURNAROUND TIME

Once a process is entered into the system, ATAT determines the average time required for it to be completed. The ATAT for a range of scheduling techniques over several process sets is listed in Table 9. AADRR works fairly well for medium and large process sets (ranging from 1 to 6 processes) with average turnaround times that are either extremely low or competitive when compared to other algorithms.

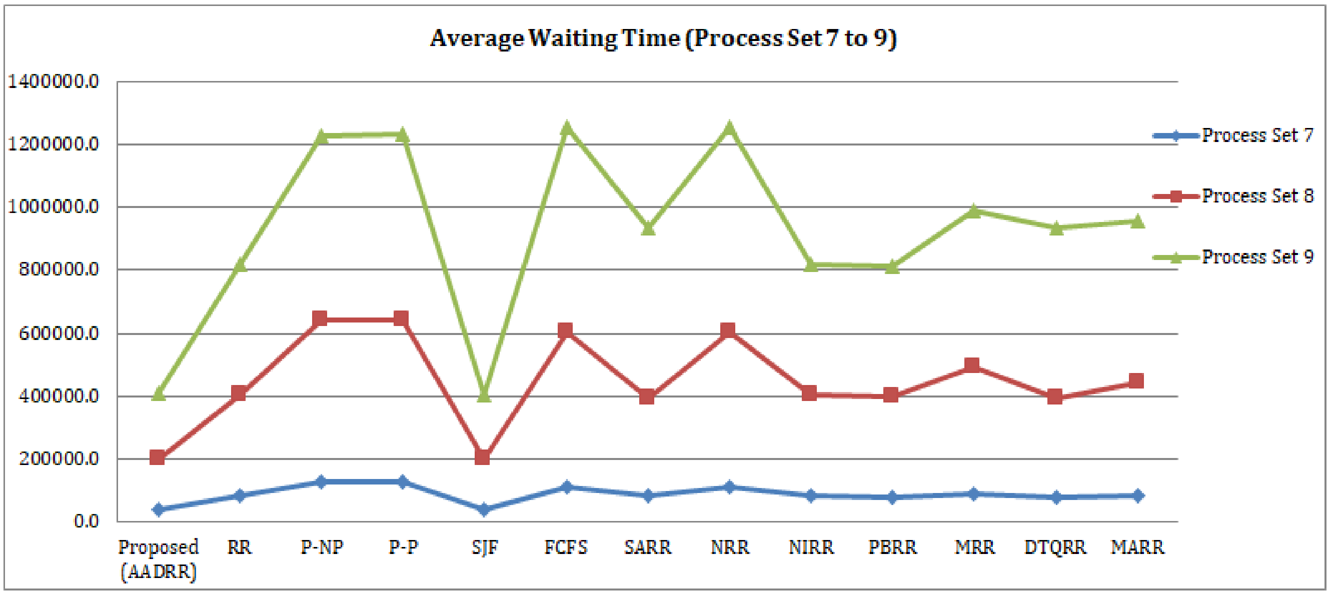

FIGURE 3. AWT for medium and large process set.

TABLE 9. A comparison of twelve (12) scheduling techniques and the proposed algorithm using ATAT.

Process Set	Proposed (AADRR)	RR	P-NP	P-P	SJF	FCFS	SARR	NRR	NIRR	PBRR	MRR	DTQRR	MARR
Process Set 1	98.50	97.25	50.00	50.00	98.5	50.00	62.5	60	92.5	142.5	111	98.5	98.5
Process Set 2	84	62.0	32.00	32.00	84.00	32.00	42.5	45	57	113	97.25	84	84
Process Set 3	81.75	69.50	35.00	35.00	81.75	35.00	35	44.25	68.75	117.75	93.25	81.75	81.75
Process Set 4	76.25	44.50	22.00	22.00	76.25	22.00	22	35	40.75	99.25	89.5	91.25	91.25
Process Set 5	27.4	31.40	31.00	31.00	38.80	30.60	30.6	32.6	25.8	43	43.2	41.20	41.20
Process Set 6	27.40	31.40	29.60	29.60	38.80	30.60	30.60	24.60	24.40	43	43.20	41.20	41.20
Process Set 7	42780.5	84282.1	125798.3	125759.6	43958.4	106926.0	82496.0	106925.0	84578.7	82938.6	90819.6	81360.7	89112.0
Process Set 8	202985.7	400640.5	637143.4	637081.5	204023.6	601257.9	391104.6	601257.9	401806.2	403260.6	493766.4	393540.4	444162.3
Process Set 9	410231.8	814020.0	1225447.1	1227802.8	409402.7	1249931.4	932214.5	1249931.4	813608.8	816160.04	991312.0	934702.9	960632.5

However, AADRR shows notably higher average turnaround times for large and medium process sets (ranging from 7 to 6 processes). It can be concluded that the effectiveness of AADRR declines with an increase in the size and complexity of process sets. AADRR is often more time-effective than most other scheduling techniques in smaller process sets (1 to 6) in so far as the turnaround times are concerned.

Fig. 4 shows a comparison of twelve (12) scheduling techniques and the proposed algorithm using ATAT for small process sets and Fig. 5 outlines a comparison of medium and large process sets. For small processes, the set (1-6), SJF, and P-NP are two simpler algorithms that frequently outperform AADRR in terms of ATAT, despite AADRR performing fairly well overall. It is a good option for smaller workloads because it can still compete with more complex algorithms like PBRR, MRR, and DTQRR.

For medium and large process sets (7-9), when compared to more conventional algorithms such as RR, FCFS, and priority-based approaches (P-NP, P-P), AADRR performs noticeably better. Even as process sets get more complex, they can retain competitive ATAT; however, SJF occasionally outperforms it. All things considered, most complex scheduling algorithms are not as effective for medium and large process sets as AADRR.

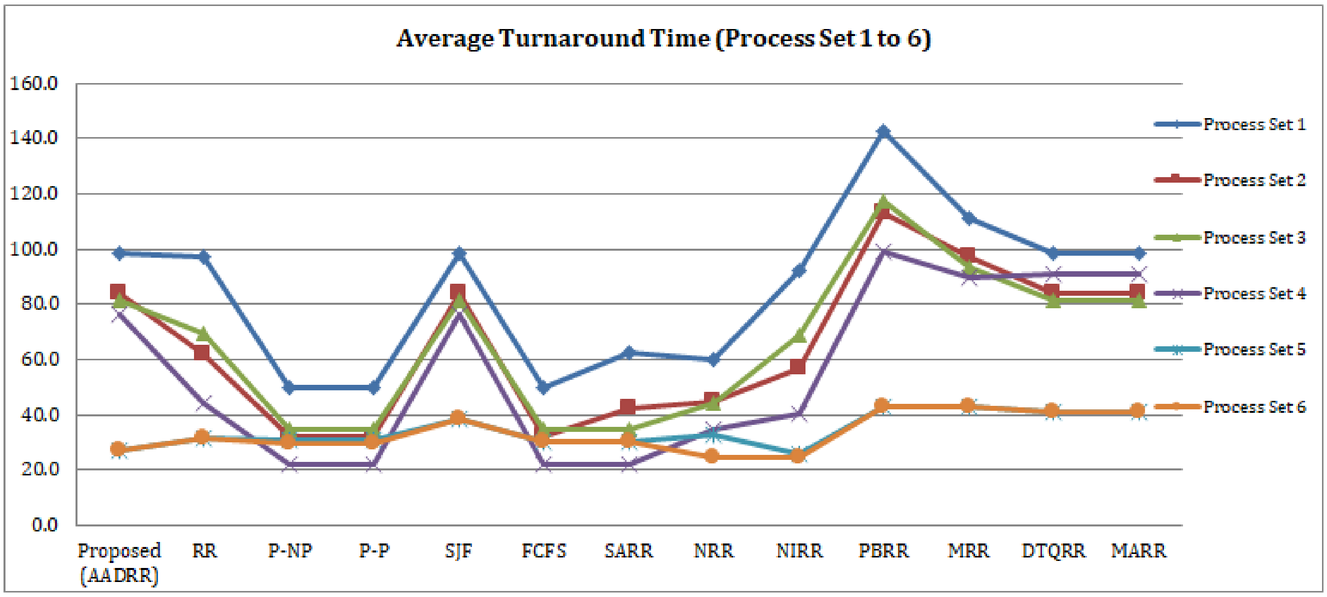

FIGURE 4. ATAT for small process set.

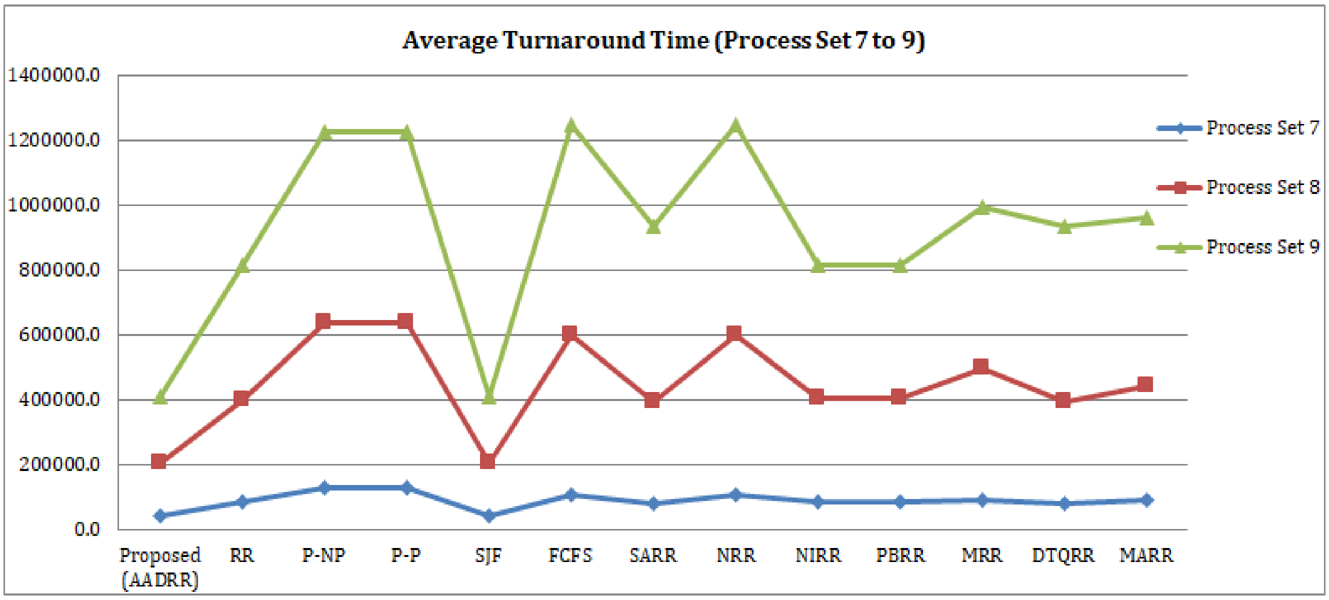

FIGURE 5. ATAT for medium and large process set.

AADRR may not scale well to larger datasets or more complex scheduling scenarios due to its dynamic nature compared to other algorithms such as FCFS, SJF, or PBRR. Different workloads may function more efficiently if the time slices allotted to each operation are modified using the AADRR approach by predetermined criteria. However, the findings imply that more expansive or intricate landscapes could not gain as much from this strategy. While AADRR may not be able to manage a larger number of processes and seems more suited for smaller or simpler workloads, it can reach turnaround times that are competitive and, in certain situations, faster than most traditional algorithms.

As the scope of the task increases, the time required for AADRR increases significantly. Rather than a simple machine-related issue, overhead costs and machine performance as production volume increases may be the primary issue. The adaptability of the method makes it easier to manage many processes or activities, each with time constraints, in dynamic or complex planning contexts. However, the increased complexity of dynamically changing processes

TABLE 10. A comparison of twelve (12) scheduling techniques and the proposed algorithm using ART.

Process Set	Proposed (AADRR)	RR	P-NP	P-P	SJF	FCFS	SARR	NRR	NIRR	PBRR	MRR	DTQRR	MARR
Process Set 1	48.5	50.00	4.50	50.00	50.00	48.5	50.00	47.5	45	7.5	4.5	46	48.5
Process Set 2	30.5	32.00	4.50	32.00	32.00	30.5	32.00	25	27.5	7.5	4.5	26.25	30.5
Process Set 3	35.75	35.00	2.00	35.00	35.0000	35.75	35.0000	35	26.75	4.25	2.75	29.75	35.75
Process Set 4	22.75	22.00	1.500	22.00	22.0000	22.75	22.00	22	17.5	3.75	2.25	18.5	20.25
Process Set 5	14.8	30.600	6.00	31.00	31.0000	26.2	30.6000	30.6	16	9.8	6	17	28.6
Process Set 6	14.8	30.600	6.00	29.60	29.60	26.2	30.6000	30.6	17	9.8	6	17	28.6
Process Set 7	40237.8	106926.0	99.00	125798.3	122023.6	41415.728	106926.00	15522.0	106925.0	152.8	129.28	55450.3	15171.4
Process Set 8	200550.8	601257.9	498.0	637143.41	633496.8	201588.7	601257.9	69450.0	601257.9	775.3	748.5	298281.9	69451.0
Process Set 9	407743.41	1249931.4	999	1225447.1	1224303.9	406914.3	1249931.4	396610.4	1249931.4	1554.09	1498.5	589678.5	396610.4

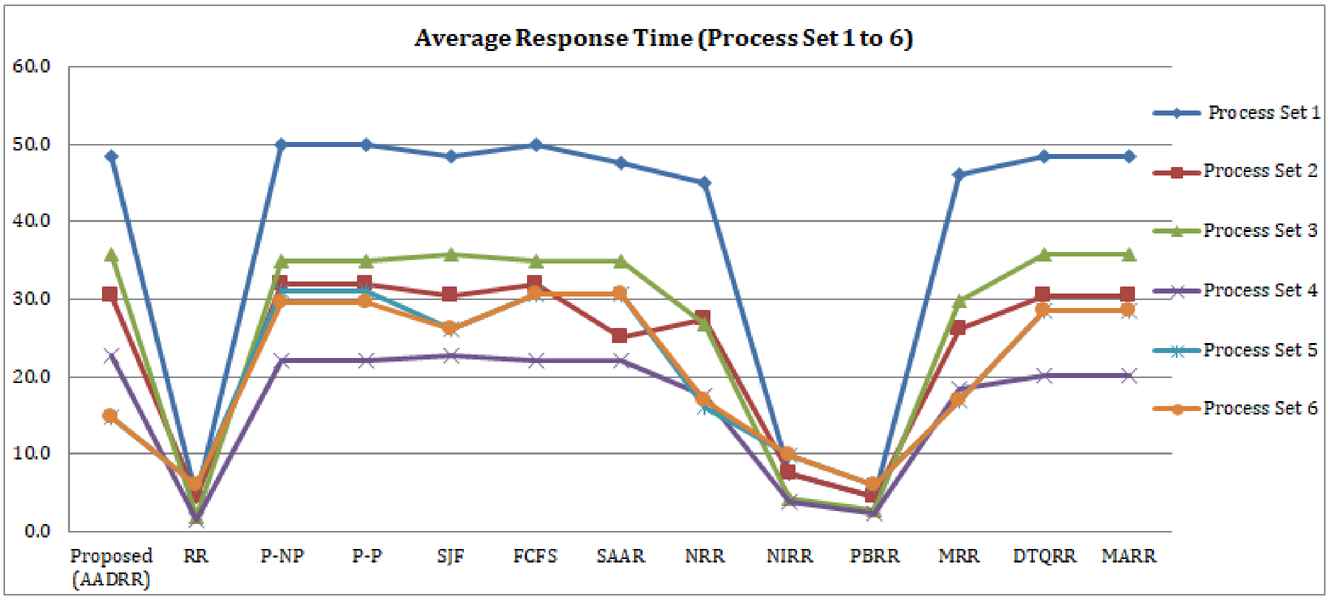

FIGURE 6. ATAT for small process set.

may cause performance issues in more time-constrained scenarios.

1) PERFORMANCE ANALYSIS OF SCHEDULING ALGORITHMS BASED ON ATAT

- **Best Performance:** In small process sets (1 to 6), the fastest turnaround times are shown by P-NP and FCFS. The simple scheduling techniques used by these algorithms make them efficient in smaller, P-NP and FCFS, indicating the quickest turnaround times. These algorithms operate well in smaller, simpler process sets because they employ straightforward scheduling strategies. Additionally, SJF and AADRR work effectively and provide comparable turnaround times, especially for smaller process sets. In large and medium Process Sets (7 to 9), compared to the other algorithms, SJF and AADRR perform noticeably better, particularly when working with big process sets. In situations where there is a lot of work, they can prioritize shorter tasks or dynamically modify their scheduling strategies to reduce turnaround times.
- **Medium Performance:** In all process sets, the performance of RR (Round Robin), SAAR, and NRR is moderate. In all process sets, the performance of RR (Round Robin), SAAR, and NRR is moderate. Even though they can handle a reasonable number of

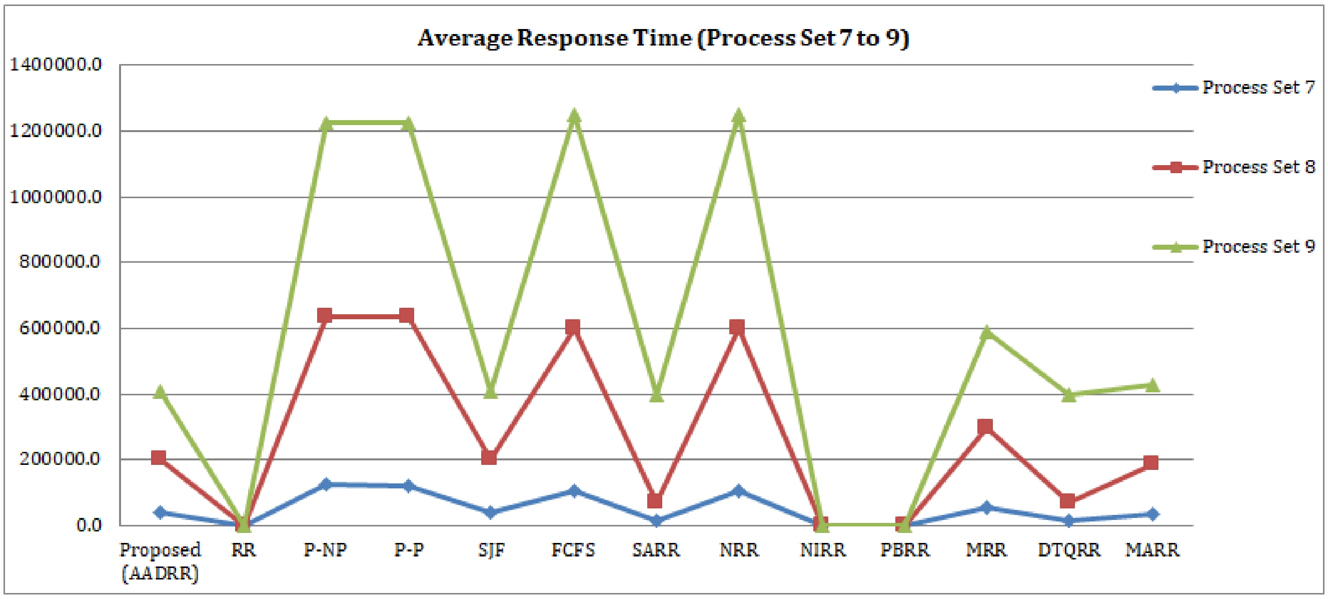

FIGURE 7. ART for medium and large process set.

processes adequately, they perform badly when the number of process sets increases due to context-switching overhead workload distribution.

- **Worst Performance:** For larger process sets, P-P, NIR, and MRR provide consistently high average turnaround times. When confronted with intricate task priorities or substantial workloads, these algorithms struggle to maintain turnaround times within acceptable bounds.

2) RECOMMENDATIONS FOR AADRR

AADRR consistently has competitive average turnaround times, particularly for bigger and more complicated process sets (7 to 9). Although it does not always have the lowest turnaround times in smaller process sets, its performance in bigger sets shows its scalability and adaptability, making it a fantastic alternative for environments that require dynamic and efficient process management. The AADRR algorithm balances fairness and efficiency by automatically adjusting time slices according to process priorities and workloads. This ensures fair resource distribution and reduces turnaround times, as seen in the results for all process sets. By effectively managing both small and large workloads, AADRR shows its capability to handle various and dynamic scheduling scenarios. Since AADRR offers balanced performance across all process sets and is remarkably efficient in large-scale scenarios, it is often highly recommended for diversified and distributed computing systems.

E. AVERAGE RESPONSE TIME

The ART for various scheduling techniques across different process sets is shown in Table 10. The time elapsed between submitting a process and having it executed at the earliest time is called response time. Lower response times indicate better performance, as the system responds to requests more quickly. While AADRR displays competitive response times in smaller process sets (ranging from 1 to 6 processes), other algorithms may perform better or handle higher loads more effectively in medium and large process sets (ranging from 7 to 9 processes). For example, the exceptionally low response times (25.0 and 22.0, respectively) in Process Sets 2 and 4 show effective handling by the AADRR algorithm in these cases.

Fig. 6 displays a comparison of twelve (12) scheduling techniques and the proposed algorithm using ART for a small process set, and Fig. 7 shows a comparison for medium and large process sets. In contrast to the smaller process sets, the process set 9 exhibits a significantly higher response time of 396610. 429. This suggests that very big or complex process sets may be difficult for the AADRR algorithm to handle, which would result in significantly longer response times.

1) PERFORMANCE ANALYSIS OF SCHEDULING ALGORITHMS: ART

- **Best Performance:** RR consistently delivers the lowest average response times across all process sets, especially in smaller sets (1 to 6). This is due to its time-slicing approach, which ensures that every process gets CPU time quickly, reducing the initial wait. PBRR also performs well, closely matching RR in small process sets. This combination of priority and round-robin principles helps balance between rapid responses and priority handling.

- **Medium Performance:** Algorithms such as SARR, NRR, and MRR offer moderate response times, particularly in smaller and midsized process sets. They outperform standard RR in some scenarios but generally fall behind in larger process sets due to more complex scheduling mechanisms. SJF and AADRR also show medium performance. While not as fast as RR in terms of response time, they provide a good balance between response time and overall efficiency.

- **Worst Performance:** FCFS, P-NP, and P-P consistently have the highest response times, especially in large process sets (7 to 9). For processes that are lower priority or not at the front of the queue, their strict scheduling causes significant delays. Additionally, MARR and DTQRR perform poorly in large process sets due to their inability to handle the added complexity effectively.

2) RECOMMENDATIONS FOR AADRR

AADRR efficiency is satisfactory for smaller and midsized process sets, but this efficiency declines as the process set size increases. While it competes well with other dynamic algorithms such as SAAR and MRR, it does not achieve the shortest response times compared to these alternatives. For applications where both response time and overall efficiency are important, AADRR is effective in small, well-defined processes.The AADRR algorithm balances fairness and efficiency by automatically changing time slices according to process priorities and workloads. This ensures that resources are shared fairly while keeping performance strong across different groups of processes. AADRR works well with smaller and medium workloads, as shown by its fast response times in Process Sets 1 to 6. But when processes become more complex, the extra work needed for these adjustments can slow down response times in larger sets. However, AADRR may not be the best option in scenarios where near-zero response times are desired; RR or PBRR would be more appropriate. AADRR can be implemented successfully in almost any computer environment due to its reasonable efficiency in meeting dynamic performance needs.

V. CONCLUSION

This research proposes a new algorithm for job scheduling known as Agent-based Adaptive Dynamic Round Robin (AADRR). AADRR works the process priority-based ranking mechanism and dynamic time quantum. Ranking is determined in real-time during the execution of processes using a weighted formula that calculates CPU burst time and priority. A software agent using the AADRR algorithm is adapted to configure the dynamic weights depending on real-time workload analysis and user preferences. AADRR allows process management in a better way and adaptability. The average burst duration is the base of the dynamic time quantum, enabling AADRR to adapt continuously to process set characteristics, ensuring efficiency and fairness across short and long tasks while reducing unnecessary preemptions.

The synthetic workload traces are used through Monte Carlo probability distribution. Many experimental results have shown that AADRR is better than traditional CPU scheduling algorithms and a few innovative CPU scheduling techniques, demonstrated excellent performance regarding ATAT and AWT. It contributes significantly to job scheduling due to its dynamic fine-tuning to changing workload traces and user-defined priorities. In summary, AADRR offers a more flexible and effective solution for various computing environments, marking an important advancement in CPU scheduling. This study looks at AADRR in a separate simulation environment.

For large workloads, AADRR has demonstrated excellent performance and is highly recommended for use in grid, edge and cloud computing environments. Future research will explore AADRR scalability in distributed systems, its applications to periodic task scheduling, and will also assess AADRR efficacy under deadline constraints, examining metrics such as preemption rate, overhead, and missed deadlines. These further assessments aim to confirm AADRR flexibility and effectiveness in computing environments with short deadlines.

COMPETING INTEREST DECLARATION

The authors certify that this research was conducted in good faith, the results are accurate and no external factors (such as personal or financial interests) have affected its scope and objective.

SUPPLEMENTARY INFORMATION

Additional files for this article are included in the form of tables, Results (both Word documents and Excel sheets), IEEE Access LaTeX Files, and image files, which offer more information and analysis to support the primary findings. As supplemental material, these files are submitted.

REFERENCES

[1] M. González-Rodríguez, L. Otero-Cerdeira, E. González-Rufino, and F. J. Rodríguez-Martínez, "Study and evaluation of CPU scheduling algorithms," *Heliyon*, vol. 10, no. 9, May 2024, Art. no. e29959.

[2] J. Bhatia, S. Mathuria, V. M. Ladwani, and S. Padmanabhan, "Optimised round Robin with virtual runtime for CPU scheduling," in *Proc. Sci. Inf. Conf.* Springer, 2024, pp. 1–17.

[3] M. Iqbal, Z. Ullah, I. A. Khan, S. Aslam, H. Shaheer, M. Humayon, M. A. Salahuddin, and A. Mehmood, "Optimizing task execution: The impact of dynamic time quantum and priorities on round Robin scheduling," *Future Internet*, vol. 15, no. 3, p. 104, Mar. 2023.

[4] A. A. Alsulami, Q. A. Al-Haija, M. I. Thanoon, and Q. Mao, "Performance evaluation of dynamic round Robin algorithms for CPU scheduling," in *Proc. SoutheastCon*, Apr. 2019, pp. 1–5.

[5] P. Chitaliya, S. Kulkarni, A. Telang, D. Zaveri, A. Shah, and K. Deulkar, "Time quantum optimization in round Robin algorithm," in *Proc. Int. Conf. Netw., Multimedia Inf. Technol. (NMITCON)*, Sep. 2023, pp. 1–6.

[6] D. Biswas and M. Samsuddoha, "Determining proficient time quantum to improve the performance of round Robin scheduling algorithm," *Int. J. Modern Educ. Comput. Sci.*, vol. 11, no. 10, pp. 33–40, Oct. 2019.

[7] S. Ali, R. Alshahrani, A. Hadadi, T. Alghamdi, F. Almuhsin, and E. E. Sharawy, "A review on the cpu scheduling algorithms: Comparative study," *Int. J. Comput. Sci. Netw. Secur.*, vol. 21, no. 1, pp. 19–26, 2021.

[8] A. Y. Ahmad, "An attempt to set standards for studying and comparing the efficiency of round Robin algorithms," *J. Educ. Sci.*, vol. 32, no. 2, pp. 11–20, Jun. 2023.

[9] P. Kavitha, "Performance assessment of CPU scheduling algorithms: A scenario-based approach with FCFS, RR, and SJF," *Bioscan*, vol. 19, no. 1, pp. 61–64, 2024.

[10] R. J. Matarneh, ''Self-adjustment time quantum in round Robin algorithm depending on burst time of the now running processes,'' *Amer. J. Appl. Sci.*, vol. 6, no. 10, pp. 1831–1837, Oct. 2009.
[11] M. Park, ''Non-preemptive fixed priority scheduling of hard real-time periodic tasks,'' in *Proc. 7th Int. Conf.*, Beijing, China. Springer, May 2007, pp. 881–888.
[12] M. U. Farooq, A. Shakoor, and A. B. Siddique, ''An efficient dynamic round Robin algorithm for CPU scheduling,'' in *Proc. Int. Conf. Commun., Comput. Digit. Syst. (C-CODE)*, Mar. 2017, pp. 244–248.
[13] A. E. Evwiekpaefe, A. Ibrahim, and M. N. Musa, ''Improved shortest job first CPU scheduling algorithm,'' *Dutse J. Pure Appl. Sci. (DUJOPAS)*, vol. 8, pp. 114–127, Jan. 2022.
[14] F. Qazi, D.-E.-S. Agha, M. Naseem, S. Badar, and F. H. Khan, ''Improving round Robin scheduling with dynamic time quantum (IRRDQ),'' *J. Appl. Eng. Technol. (JAET)*, vol. 7, no. 2, pp. 70–82, Dec. 2023.
[15] S. M. Mostafa and H. Amano, ''Dynamic round Robin CPU scheduling algorithm based on K-means clustering technique,'' *Appl. Sci.*, vol. 10, no. 15, p. 5134, Jul. 2020.
[16] A. K. Gupta, N. S. Yadav, and D. Goyal, ''Design and performance evaluation of smart job first dynamic round Robin (SJFDRR) scheduling algorithm with smart time quantum,'' *Amer. Sci. Res. J. Eng., Technol., Sci. (ASRJETS)*, vol. 26, no. 4, pp. 66–78, Dec. 2016.
[17] S. Zouaoui, L. Boussaid, and A. Mtibaa, ''Priority based round Robin (PBRR) CPU scheduling algorithm,'' *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 9, no. 1, p. 190, Feb. 2019.
[18] K. Chandiramani, R. Verma, and M. Sivagami, ''A modified priority preemptive algorithm for CPU scheduling,'' *Proc. Comput. Sci.*, vol. 165, pp. 363–369, Jan. 2019.
[19] H. M. Abu-Dalbouh, ''A new combination approach to CPU scheduling based on priority and round-Robin algorithms for assigning a priority to a process and eliminating starvation,'' *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 4, 2022.
[20] P. S. Sharma, S. Kumar, M. S. Gaur, and V. Jain, ''A novel intelligent round Robin CPU scheduling algorithm,'' *Int. J. Inf. Technol.*, vol. 14, no. 3, pp. 1475–1482, May 2022.
[21] C. Sharma, S. Sharma, S. Kautish, S. A. M. Alsallami, E. M. Khalil, and A. W. Mohamed, ''A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time,'' *Alexandria Eng. J.*, vol. 61, no. 12, pp. 10527–10538, Dec. 2022.
[22] Y. Berhanu, A. Alemu, and M. Kumar, ''Dynamic time quantum based round Robin CPU scheduling algorithm,'' *Int. J. Comput. Appl.*, vol. 167, no. 13, pp. 48–55, Jun. 2017.
[23] A. Y. Ahmad, ''Improved round Robin CPU scheduling algorithm with different arrival times based on dynamic quantum,'' *J. Educ. Sci.*, vol. 31, no. 4, pp. 105–115, Dec. 2022.
[24] T. Jha and T. Choudhury, ''A hybrid of round Robin and shortest job first CPU scheduling algorithm for minimizing average waiting time,'' in *Proc. 2nd Int. Conf. Green Comput. Internet Things (ICGCIoT)*, Aug. 2018, pp. 466–470.
[25] C. Yaashuwanth and R. Ramesh, ''Design of real time scheduler simulator and development of modified round Robin architecture for real time system,'' *Int. J. Comput. Electr. Eng.*, vol. 10, no. 3, pp. 43–47, 2010.
[26] C. Yaashuwanth and R. Ramesh, ''Intelligent time slice for round Robin in real time operating systems,'' *IJRRAS*, vol. 2, no. 2, pp. 126–131, 2010.
[27] A. E. A. Agha and S. J. Jassbi, ''A new method to improve round Robin scheduling algorithm with quantum time based on harmonic-arithmetic mean (HARM),'' *Int. J. Inf. Technol. Comput. Sci.*, vol. 5, no. 7, pp. 56–62, Jun. 2013.
[28] S. Elmougy, S. Sarhan, and M. Joundy, ''A novel hybrid of shortest job first and round Robin with dynamic variable quantum time task scheduling technique,'' *J. Cloud Comput.*, vol. 6, no. 1, pp. 1–12, Dec. 2017.
[29] U. Shafi, M. Shah, A. Wahid, K. Abbasi, Q. Javaid, M. Asghar, and M. Haider, ''A novel amended dynamic round Robin scheduling algorithm for timeshared systems,'' *Int. Arab J. Inf. Technol.*, vol. 17, no. 1, pp. 90–98, Jan. 2020.
[30] M. Sohrawordi, A. Ehasn, U. Palash, and H. Mahabub, ''A modified round Robin CPU scheduling algorithm with dynamic time quantum,'' *Int. J. Adv. Res.*, vol. 7, no. 2, pp. 422–429, 2019.
[31] V. Salmani, S. T. Zargar, and M. Naghibzadeh, ''A modified maximum urgency first scheduling algorithm for real-time tasks,'' *World Acad. Sci., Eng. Technol.*, Jan. 2005.

ZAFAR IQBAL KHAN received the B.S. degree in computer science from Dr. A. Q. Khan Institute of Computer Sciences and Information Technology (KICSIT), Kahuta, Pakistan, and the M.S. degree in software engineering from International Islamic University (IIU), Islamabad, Pakistan, in 2009. He is currently pursuing the Ph.D. degree with the Department of Software Engineering, National University of Modern Languages, Islamabad. He is an Assistant Professor with the Department of Computer Science, Institute of Space Technology, KICSIT Kahuta Campus, Islamabad. He has 17 years of university teaching experience and 20 years of experience in software development. His current research interests focus on developing multi-level agent job scheduling algorithms, software quality engineering, cloud computing, software engineering, requirements engineering, and web engineering.

MUZAFAR KHAN received the M.Sc. degree in software engineering from the Blekinge Institute of Technology, Sweden, in 2008, and the Ph.D. degree in human–computer interaction from the Universiti Teknologi PETRONAS, Malaysia, in 2012. Currently, he is the Head of Department (HoD) of software engineering with the National University of Modern Languages, Rawalpindi, Pakistan. He is passionate about interdisciplinary research, particularly in the areas of human–computer interaction (HCI), cloud job scheduling, and software engineering.

SYED NASIR MEHMOOD SHAH received the Ph.D. degree in information technology from Universiti Teknologi PETRONAS, Malaysia, in 2012. He is currently the Head of Academics and a Full Professor with the Institute of Space Technology, KICSIT Kahuta Campus, Islamabad, Pakistan. He has published 47 research papers in peer-reviewed conferences, quality journals, and book chapters. His research interests include cloud computing, cybersecurity, distributed computing, grid computing, operating systems, parallel and distributed processing techniques and algorithms, software engineering, and software quality assurance.