Vol. 7, No. 2 | July – December 2023

Dynamic Time Quantum Computation for Improved Round Robin Scheduling Algorithm Using Quartiles and Randomization (IRRQR)

Asif Yar[1], Bushra Jamil[1*], Humaira Ijaz[1]

Abstract:

Scheduling is a decision-making process through which large numbers of tasks compete for various system resources. The availability of limited resources makes scheduling a challenge. Among resources, the processoris the most important resource for in-time completion of tasks therefore; developing an efficient processor scheduler is still a topic of interest. We have applied a statistical approach that combines concept of median quartiles, upper quartiles and randomness to adjust the time quantum for each process with the objective of optimizing the allocation of CPU time to a set of processes. We have considered average waiting time, average turnaround time, and the number of context switches as performance metrics and compared our algorithm (IRRQR) with nine other dynamic time quantum adjustment algorithms. The comparisons with these algorithms reveal the effectiveness of IRRQR in terms of minimizing number of contexts switches up to 25%, average waiting time up to 13.7%, and average turnaround time by 8.4%.

Keywords: *Scheduling Algorithm; Round Robin; Quartiles; Time Quantum; Randomization; Average Waiting Time; Average Turnaround Time*

1. Introduction

Modern operating systems were designed with the idea of multi-programming to deal with the problem of underutilization of system resources that enabled the interleaved execution of multiple jobs on a single processor machine [1]. These multi-programmed operating systems need a plan for the specific ordering of these jobs in the ready queue according to some defined criteria to select amongst them for CPU allocation to maximize resource utilization. This process is called CPU scheduling in which a scheduler deals with ordering and selection of jobs from the ready queue which mainly concerns throughput, latency, and response time. After that, the dispatcher transfers the control of the CPU to the selected job. As the nature of multi-programming-based CPU scheduling like First Come First Serve (FCFS), and Shortest Job First (SJF), priority scheduling is non-preemptive and therefore is not well suited for I/O bound jobs and online users.

The idea of the time-sharing concept was introduced in multi-programming-based operating systems to overcome issues of non-preemption. In time-sharing systems, the CPU time is shared in short time intervals/slots among multiple jobs simultaneously by frequent context switching so that all jobs run seamlessly without any problem. This time interval is called a time quantum (TQ), time slice, or time slot. If the job completes its CPU burst before its time slice expires the job preempts out of CPU like the FCFS algorithm. However, if the job completely consumes its

[1] Department of CS & IT, University of Sargodha, Sargodha, Pakistan
Corresponding Authors: bushra.jamil@uos.edu.pk

time quantum, it preempts and moves to the tail of the ready queue. As the time quantum is usually small, switching among jobs occurs so frequently that the users are unaware of the fact that jobs are sharing system resources and can also interact with running jobs. Some examples of preemptive-based job scheduling algorithms are Shortest Remaining Time First (SRTF), Priority Scheduling, and Round Robin (RR).

Round Robin (RR) is the most widely used preemptive scheduling algorithm for a time-sharing environment [2]. In RR a fixed small static time quantum (TQ) is defined for which the dispatcher transfers CPU control to each job in the ready queue in FCFS order [3]. The selection of TQ greatly affects the performance of the RR scheduling algorithm. The selection of small-time quantum results in an increased number of Context Switches that increase the overhead as the CPU does nothing except save and restore the context of jobs. Whereas, a large quantum leads to increased waiting time causing RR to behave as FCFS (First Come First Serve) [4]. Increasing the context switching increases the overhead as the CPU does nothing except save and restore the context of jobs.

Dynamic time quantum selection dynamically adjusts the time quantum for each process based on its behavior. It not only gives a fair share of CPU time to every process by prioritizing them but also improves system response time along with optimal utilization of resources. In the field of operating systems dynamic time quantum selection is an active area of research to improve the performance of system. Therefore, there is need to develop a dynamic time quantum selection techniques that optimizes CPU time allocation to different processes minimizing turnaround time along fare share to each process. In the field of operating systems dynamic time quantum selection is an active area of research to improve the performance of system. Some researchers have already proposed a modified version of Round Robin, in which dynamic time quantum is continuously computed for currently running jobs based on burst times of currently running jobs [5]. However, existing algorithms are still not efficient enough in reducing the number of context switches (CS), average waiting time (AWT), and average turnaround time (ATT). To achieve these objectives, we propose a dynamic time quantum computing algorithm, for an improved Round Robin Scheduling algorithm that calculates TQ dynamically based on Quartile and Randomization (IRRQR) of the remaining burst time of jobs in the ready queue. The median quartile is used to group processes into short CPU bursts and long CPU bursts for adjusting time quantum to shorter for processes with short CPU burst lengths and longer time quantum for long CPU bursts. Whereas the upper quartile is used as a threshold for long-running processes in which the longer time quantum is assigned to processes with CPU bursts above the upper quartile and shorter time quantum with CPU bursts below upper quartile. Afterwards, randomization is used to select any random value from the ready queue. The quartiles are used to group processes with similar CPU burst lengths and assign them similar time quantum values, while randomization can be useful in preventing long-running processes from monopolizing the CPU.

Therefore, we have used a combination of median quartile, upper quartile, and randomness in dynamic time quantum selection that provides a fair share of CPU time to every process to ensure efficient allocation of the CPU time to different processes, preventing long-running processes from monopolizing the CPU. The objectives of this algorithm are to minimize the total number of context switches, average turnaround time, and average waiting time.

The rest of this paper is organized in the following order. Related work is described in Section 2. The design and implementation of the proposed algorithm is presented in Section

3. Experimental results are discussed in Section 4. We conclude and describe the future work in section 5.

2. Related Work

In this section, we review some existing modified Round Robin Scheduling algorithms that have been proposed till now for enhancing the performance of the classical RR algorithm by computing dynamic time quantum. In [5], Gowda proposes a statistical approach for dynamic time quantum. They divide jobs into four categories based on burst time and time quantum. For each category min-max spread size is used. They compute dynamic time quantum by taking the square root of the multiple of a total number of jobs, mean, and standard deviation. In [6], Mishra et al. proposed dynamic time quantum computation by sorting jobs in ascending order of their burst times. Then, the time quantum is selected equal to the burst time of the first job. An amended Dynamic Round Robin (ADRR) [7] is proposed by Shafi et al. in which they arrange all jobs in increased order of their burst times and the time quantum is set to the lowest burst time. In each cycle, if the time quantum is less than 20, set the time quantum to 20. If the remaining burst time of a current job is less than half of the time quantum, let this job complete its execution. Therefore, a job with minimum burst waits for minimum time. After each cycle, the quantum is readjusted.

The technique suggested in [8] by Berhanu et al. is to initially sort the jobs in increasing order of their burst times and after that set the quantum equal to the burst of the first job and if the remaining burst time of a currently running job is less than time quantum, assign CPU to same job again till it completes its execution.

In [9], LaxmiJeevani et al. suggest the calculation of dynamic time quantum, firstly by fixing quantum to k unit of time statically and then assigning CPU for that time quantum to the first coming job from the queue. After that, the time quantum is set to the burst time of the job with the lowest burst time present in the ready queue. Tajwar et. al. in [10], propose to arrange jobs according to their burst time in ascending order and set a time quantum equal to the average of these jobs, then assign CPU to each job equal to that time quantum.

In [11], Kumar et al. propose an algorithm that sorts the jobs in ascending order and computes dynamic time quantum by calculating the harmonic mean of jobs that have arrived in the ready queue. According to [12], Ranjan et al. dynamic time quantum is calculated by summing up the burst time of all jobs in the ready queue and dividing that sum by the total number of jobs, which is called arithmetic mean. Emami in [13] proposes a harmonic-arithmetic mean (HARM) algorithm for dynamic time quantum. The quantum is set to the Harmonic mean of the burst time of all the jobs if some jobs have greatly larger burst times than other jobs and jobs are heterogeneous then set time quantum to the arithmetic mean of the burst time of jobs.

In [14], Mohanty et al. propose to compute dynamic time quantum by sorting jobs in ascending order, calculating the median of the burst time of a job that has arrived in the ready queue. Another technique is presented in [15] by Matarneh, in which jobs are sorted in ascending order to calculate the median, and the time quantum is set to the median. If the quantum is less than the threshold value of 25, it is set equal to the threshold value. In [16], Nayak et al. computes the dynamic time quantum by sorting jobs in ascending order and finding median and highest burst time. They fixed the time quantum to an average of the median and the highest burst time.

Varma et al. in [17], arrange jobs in increased order to their burst times and calculate time quantum by taking the square root of the sum of squares of burst time divided by the total number of jobs, also called root

mean square. In [18], Khokhar et al. propose a dynamic time quantum technique using an average of mean and median. Allocate CPU to each job and if the remaining burst time of a current job is less or equal to the burst time, allocate CPU again to it till it completes its execution. In [19], Zhang et al., suggest dynamic time quantum computation based on median theory. In another experimental study, Iqbal and fellows compared four variations of RR with conventional RR [20]. The study reveals that variations of RR perform better than conventional RR.

Najafi and Samira propose a method to calculate dynamic time quantum using machine learning [21]. They use several processes and their average, maximum, and minimum burst times are used as features for training the data set. After training ML classifier predicts the time quantum. Vayadande et al. propose a method to calculate dynamic time quantum for RR. In this method, the dynamic TQ for each cycle is calculated using a prescribed formula [22].

Sakshi and fellows present an algorithm to calculate dynamic time quantum using the median and average burst time of every process. The proposed algorithm is compared with four other state-of-the-art algorithms to reveal the superiority of the proposed algorithm [23]. In another study, the priority of the process and RR scheduling algorithm are combined to calculate the dynamic time quantum to take advantage of both algorithms [24].

Table 1 presents the summary of the existing dynamic time quantum computation techniques used in different studies.

TABLE I**.** *Comparison of*

Sr. #	**Tech.**	**Short Comings**	**Ref.**
1	Statistical Method		[5]
2	Sorting	Unfair CPU allocation due to prioritization.	[6,7,8 ,9,10]
3	Harmonic mean	It gives priority to smaller values causing more context switches	[11]
4	Arithmetic mean	It assumes that the distribution of time quantum is normal so not suitable for real-world cases.	[12]
5	Median	Not suitable for systems with highly variable workloads	[14,15,16, 19, 23]
6	Root mean Square	Calculating RMS is time-consuming	[17]
7	Mean and median	Needs more computations to result in long delays.	[18]
8	ML	May not be accurate for future workloads	[21]

The value of time quantum can greatly affect the performance of the RR algorithm, and existing algorithms are still not efficient enough in reducing the number of context switches, average waiting time, and average turnaround time. To achieve these purposes, we implement a variant of Round Robin Scheduling Algorithm using Quartiles and Randomization (IRRQR).

3. Design and Implementation

Static time quantum is a limitation of the RR algorithm that degrades its performance. Therefore, in this work, we propose a methodology for dynamic time quantum computation that would be computed in each cycle of the RR algorithm. Our dynamic time quantum results in:

1. Minimizing the number of context switches

2. Minimizing average waiting time
3. Minimizing average turnaround time

For dynamic time quantum computation, we are using the concepts of median quartile (MQ), upper quartile (UQ), and randomization. For MQ we compute the median of the total number of jobs, while for UQ median of the second half of jobs is computed. We compute these medians using equation 1.

$$Median = \begin{cases} E(\frac{N}{2}) & \text{if N is odd} \\ \frac{E(\frac{N}{2})+E(\frac{N}{2}-1)}{2} & \text{otherwise} \end{cases} \quad (1)$$

Where E(N) indicates the element at index x and N is the total number of jobs. After MQ and UQ are computed, the time quantum is selected randomly from both computed values in each cycle of the RR algorithm. Therefore, at least 50 percent of the jobs will complete their execution in the first round of the algorithm.

We present our proposed algorithm IRRQR as follows.

Algorithm 1 IRRQR Algorithm for Task Scheduling

INPUT: RQ ($P1, P2, \ldots PN$)
OUTPUT: AWT, ATT

```
do
    Sort RQ in ascending according to BT of jobs
    Compute M_Q
    Compute U_Q
    TQ = Rand(M_Q , U_Q)
    for ( i ← 1 to N ) do
        Assign TQ to Job P_i
        if BT_i ≤ TQ then
            remove P_i from RQ
        else
            BT_i = BT_i - TQ
            RQ ⟸ P_i
        end if
        Compute WT_i for all jobs in RQ
    end for
while RQ!=∅
calculate AWT,ATT
```

The symbols used in the IRRQR algorithm are shown in Table 2.

TABLE II. Algorithm symbols

Symbol	Meaning
RQ	Ready Queue of N Jobs
AWT	Average waiting time
ATT	Average turnaround time
MQ	Median of burst time of all jobs RQ
UQ	Quartile of jobs in the second half of RQ
TQ	Time quantum
WT	Waiting time

In IRRQR, initially, jobs in the ready queue are ordered in ascending order according to their burst times. Then, we compute MQ and UQ. TQ is calculated as a random number between MQ and UQ. This TQ is assigned to every job in the current iteration of the RR algorithm. If the job completes its execution and terminates, it is removed from the ready queue otherwise, it is put back at the tail of the ready queue. This whole job continues until the ready queue becomes empty.

3.1 Illustration

We illustrate the performance of our proposed algorithm using a simple scenario. Let us consider there are five jobs P0, P1, P2, P3, and P4 in a ready queue. The arrival time for each job is zero while the burst time is 48, 22, 70, 74, 10. According to our proposed methodology, jobs in the ready queue will be arranged in ascending based on their burst times and the new sequence will be P4, P1, P0, P2, P3. Random time quantum between the median quartile and upper quartile is calculated i.e. TQ = 70. CPU will be allocated to each job and in the first iteration of the algorithm time, the quantum value will be 70, so jobs P4, P1, P0, and P2 will complete their execution as their remaining burst time is zero and will be taken out of the ready queue. The remaining burst time of job P3 will be 4. In the second iteration of the algorithm TQ = 4 as there is a single job in the ready queue. So, CPU will be allocated to job P3 when job P3

terminates ready queue will be empty. The average turnaround is 99.2 units and the average waiting time is 54.4 units.

3.2 Complexity Analysis

The time complexity of the presented algorithm depends on the length of the ready queue that is N. As the scheduler sorts this queue, therefore, the time complexity of sorting is O(nlogn). Computation of UQ, MQ, and TQ takes O(1) time. TQ has to be assigned to every job so it will take O(n) time. As a result, the time complexity of the IRRQR algorithm is O(nlogn) time while the space complexity is O(n).

4. Performance Evaluation

In this section, we present the results of our proposed IRRQR algorithm for different scenarios against existing state-of-the-art algorithms. These algorithms are: Round Robin (RR) [4], Improved Round Robin Variant Quantum (IRRVQ) [6], Amended Dynamic Round Robin (ADRR) [7], Dynamic Time Quantum Round Robin (DTQRR) [8], Round Robin Based Effective Time Slice (RRBETC) [10], SubContrary Mean Dynamic Round Robin (SMDRR) [11], Shortest Remaining Burst Round Robin (SRBRR) [14], Self Adjustment Time Quantum Round Robin (SATQRR) [15], Improved Round Robin (IRR) [16], Improved Shortest Remaining Burst Round Robin(ISRBRR) [17] and Median Based Round Robin (MBRR) [18].

For comparison, we select three metrics which are the number of average waiting time, average turnaround time and no of context switches.

Average Waiting Time(AWT): is the average waiting time of all the jobs that can be calculated using equation 2.

$$\text{AWT} = \sum_{1=1}^{m} WT_i \qquad (2)$$

Where m is the number of jobs in the ready queue. The waiting time for each job can be computed using Equation 3.

$$WT_i = CT_i + BT_i + AT_i \quad \forall i \in T \quad (3)$$

Average Turnaround Time (ATT): the average turn time of all the jobs in the ready queue can be computed usingequation 4.

$$\text{ATT} = \sum_{1=1}^{n} TT_i \qquad (4)$$

Turnaround time can be computed using equation 5.

$$TT_i = CT_i - AT_i \quad \forall i \in T \quad (5)$$

Where CT_i is the completion time and AT_i is the arrival time of i^{th} job.

1) Assumptions

We assume that all the jobs in the ready queue have equal priority. All the jobs are independent of each other. The arrival time and burst time of all jobs are known in advance. All jobs submitted for execution are CPU-bound. Forall scenarios, we have taken five jobs. We have not considered the overhead of context switching also sorting timefor jobs is considered zero.

2) Simulation Environment

To evaluate the performance of the proposed IRRQR scheduling algorithm, we have performed simulations. These simulations are carried out in Scala 2.11.8 on the Ubuntu 16.04 operating system, on a 4-core system with 4GB RAM and a 2.67GHz processor. In the next section, numerical and simulation results are discussed.

3) Results

In this section, we present six cases and compare our proposed algorithm results with the traditional round robin algorithm as well as eight other existing algorithms proposed by different researchers. For each case, we take seven jobs with different burst times. We divide six cases into two categories.

1. Zero arrival time

2. Heterogeneous arrival times

In each category, there are three different cases given below.

1. **First case**: All jobs are heterogeneous means they have burst times of different lengths.
2. **Second case**: One of the jobs has less burst time.
3. **Third case**: One of the jobs has more burst time.

4.3.1Zero Arrival Time

Case 1: Burst Times in ascending order In this case, a ready queue with seven identical jobs is considered where jobs are in increasing order of their burst time. Table 3 shows the burst time of each job.

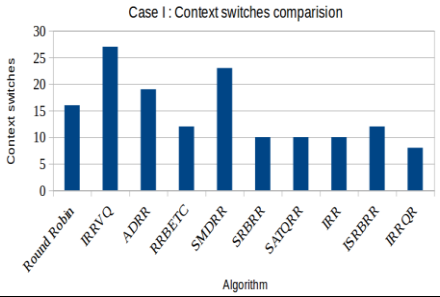
TABLE III. Burst Times in ascending order

Job	P1	P2	P3	P4	P5	P6	P7
Burst Time	5	23	34	41	66	78	80

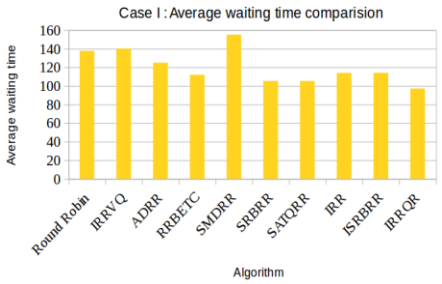
Table 4 presents the comparison results of algorithms.

TABLE IV. Comparison Results

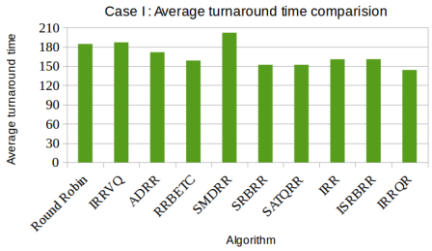
Algorithm	**CS**	**AWT**	**ATT**
Round Robin	16	137.7	184.4
IRRVQ	27	140.1	186.9
ADRR	19	124.9	171.6
RRBETC	12	111.9	158.6
SMDRR	23	155.1	201.9
SRBRR	10	105.3	152
SATQRR	10	105.3	152
IRR	10	113.9	160.6
ISRBRR	12	114	160.7
IRRQR	8	97.1	143.9



(a) Context Switches



(b) Average Waiting Time



(c) Average Turnaround Time

Fig I: Comparison Results of Burst Times in Ascending Order

Figure 1 shows the comparison of selected algorithms with the proposed algorithm. Algorithms are shown along the x-axis while the y-axis presents no of CS, AWT and ATT in figures 1a, 1b, and 1c respectively. The figure presents that our IRRQR algorithm gives the least no of CS, AWT, and ATT as compared to all existing algorithms while SRBRR and SATQRR give competitive results. The results show that the CS, AWT, and ATT reduce by 25%, 8.4%, and 5.6% respectively in comparison with other best approaches.

Case 2: Burst Times in descending order In this case, a ready queue with seven jobs has been considered where jobs have burst times in descending order as shown in Table 5.

TABLE V. Burst Times in descending order

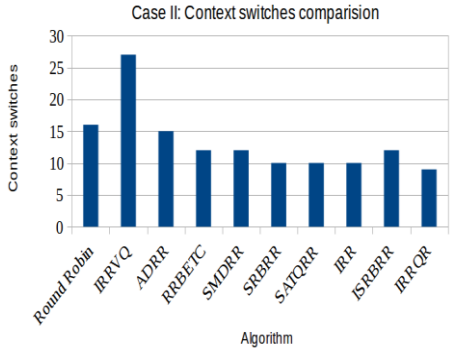
Job	P1	P2	P3	P4	P5	P6	P7
Burst Time	94	89	79	60	57	52	50

Table 6 presents the comparison results of algorithms.

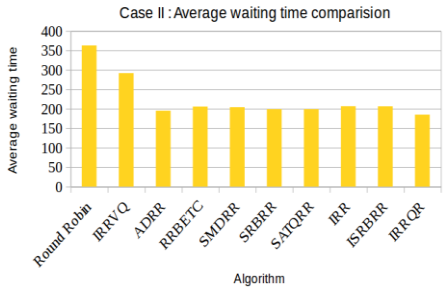
TABLE VI. Comparison Results

Algorithm	CS	AWT	ATT
Round Robin	16	362.9	431.6
IRRVQ	27	291.8	360.6
ADRR	15	195	263.7
RRBETC	12	205.7	274.4
SMDRR	12	204.2	273
SRBRR	10	199.2	268
SATQRR	10	199.2	268
IRR	10	206.6	275.3
ISRBRR	12	206.4	275.1
IRRQR	9	185	253.8

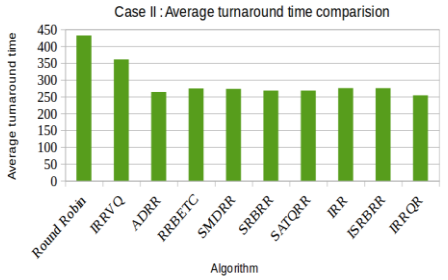
Figure 2 shows the comparison of selected algorithms with the IRRQR algorithm. Algorithms are shown along the x-axis, while the y-axis presents number of CS, AWT, and ATT in figures 2a, 2b, and 2c respectively. The figure presents that the presented IRRQR algorithm gives the least no of CS, AWT, and ATT as compared to all existing algorithms while ADRR, SRBRR, and SATQRR give competitive results. Our algorithm decreases CS, AWT, and ATT by 11.1%, 7.7%, and 5.6% respectively in comparison with SRBRR and SATQRR algorithms.



(a) Context Switches



(b) Average Waiting Time



(c) Average Turnaround Time

Fig II: Comparison Results of Burst Times in Descending Order

Case 3: Random Burst Times

In this case, seven jobs with random burst times are considered as shown in Table 7.

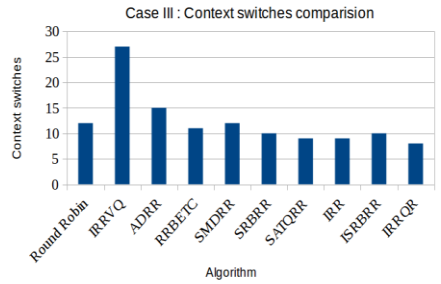
TABLE VII. Jobs in random order

Job	P1	P2	P3	P4	P5	P6	P7
Burst Time	39	20	43	26	31	42	28

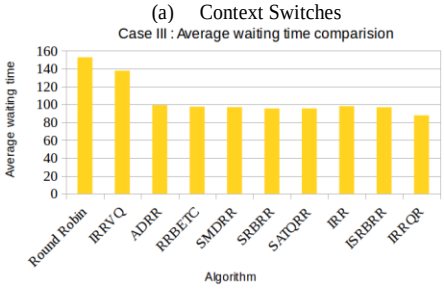
Table 8 shows the comparison results of the algorithms.

TABLE VIII. Comparison Results

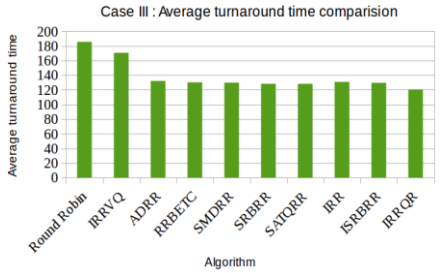
Algorithm	CS	AWT	ATT
Round Robin	12	152.7	185.4
IRRVQ	27	137.7	170.4
ADRR	15	99.2	132
RRBETC	11	97.4	130.1
SMDRR	12	96.9	129.6
SRBRR	10	95.4	128.1
SATQRR	9	95.4	128.1
IRR	9	98	130.7
ISRBRR	10	96.7	129.4
IRRQR	8	87.7	120.4



(a) Context Switches



(b) Average Waiting Time



(c) Average Turnaround Time

Fig III: Comparison Results of Random Burst Times

Figure 3 shows the comparison of selected algorithms with the proposed algorithm. Algorithms are shown alongthe x-axis while the y-axis presents no context switches, average waiting time, and average turnaround time in figures 3a, 3b, and 3c respectively. The figure presents that the IRRQR algorithm gives the least no of CS, AWT, and ATT as compared to all existing algorithms while SATQRR and SRBRR give competitive results. Our algorithm gives 12.5%, 8.7% and 6.3% reduced CS, AWT, and ATT as compared to SATQRR while 25%, 8.7%, and 6.3% least CS, AWT, and ATT in comparison with SRBRR.

4.3.2Non- Zero Arrival Time

Case 4: Burst Times in ascending order In this case, a ready queue with seven identical jobs is considered where jobs are in ascending orderof their burst time. Table 9 shows the arrival time and burst time of each job.

TABLE IX. Jobs in ascending order

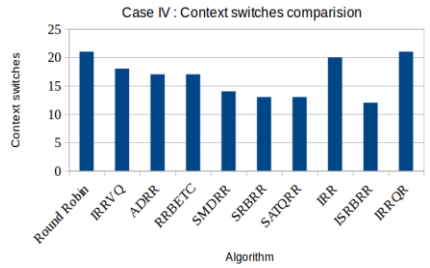
Job	P1	P2	P3	P4	P5	P6	P7
Arrival Time	4	0	7	11	19	13	23
Burst Time	7	11	38	53	61	72	74

Table 10 shows the comparison results of the IRRQR algorithm with other algorithms.

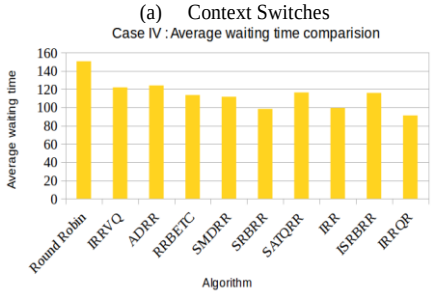
TABLE X. Comparison Results

Algorithm	**CS**	**AWT**	**ATT**
Round Robin	16	150.6	195.7
IRRVQ	21	122	167.1
ADRR	18	124.1	169.3
RRBETC	17	113.6	158.7
SMDRR	17	111.7	156.9
SRBRR	14	98.4	143.6
SATQRR	13	116.4	161.6
IRR	13	99.4	144.6
ISRBRR	20	115.9	161
IRRQR	12	91.2	136.4

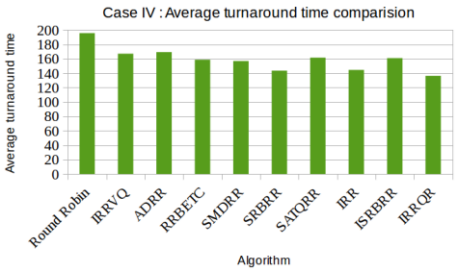
Figure 4 shows the comparison of IRRQR with other selected algorithms. Algorithms are shown along the x-axis and the y-axis presents number of CS, AWT, and ATT in figures 4a, 4b, and 4c respectively. The figure presents that the proposed IRRQR algorithm gives the least no of CS, AWT, and ATT as compared to all existing algorithms while SRBRR and IRR give competitive results. Our algorithm gives 8.3%, 8.9% and 6.01% reduced CS, AWT, and ATT as compared to the best-performing IRR algorithm.



(a) Context Switches



(b) Average Waiting Time



(c) Average Turnaround Time

Fig IV: Comparison Results with non-zero arrival time

Case 5: Burst Times in descending order

In this case, a ready queue with seven identical jobs P1, P2, P3, P4, P5, P6, and P7 has been considered where jobs are in descending order of their burst time. Table 10 shows the arrival time and burst timeof each job.

TABLE XI. Jobs in ascending order

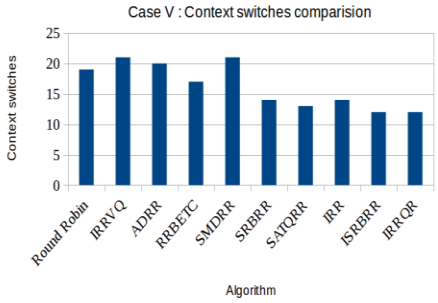
Job	P1	P2	P3	P4	P5	P6	P7
Arrival Time	11	2	7	4	16	0	9
Burst Time	103	96	85	72	46	19	7

Table 12 shows the comparison results of the algorithms

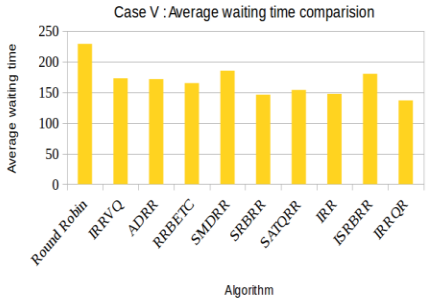
TABLE XII. Comparison Results

Algorithm	**CS**	**AWT**	**ATT**
Round Robin	19	228.9	290
IRRVQ	21	172.7	233.9
ADRR	20	171.4	232.6
RRBETC	17	164.9	226
SMDRR	21	185.1	246.3
SRBRR	14	146	207.1
SATQRR	13	153.7	214.6
IRR	14	147.3	208.4
ISRBRR	12	180.1	241.3
IRRQR	12	136.6	197.7

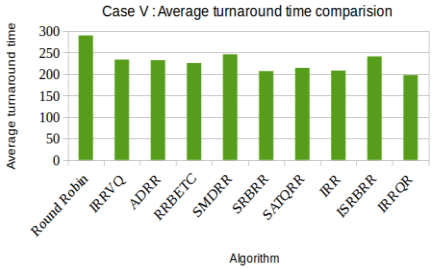
Figure 5 shows the comparison of selected algorithms with our IRRQR algorithm. Algorithms are shown alongthe x-axis while the y-axis presents no of context switches, average waiting time, and average turnaround time in figures 5a, 5b, and 5c respectively.



(a) Context Switches



(b) Average Waiting Time



(c) Average Turnaround Time

Fig V: Comparison Results with descending burst times for non-zero arrival time

The figure presents that the presented IRRQR algorithm gives the least no of CS, AWT, and ATT as compared to all existing algorithms while SRBRR and IRR give competitive results. IRRQR decreased CS, AWT, and ATT by 16.7%, 7.8%, and 5.4% respectively as compared to IRR.

Case 5: Random Burst Times In this case, a Ready queue with seven identical jobs P1, P2, P3, P4, P5, P6, and P7 has been considered where jobs are in random order of their burst time. Table 13 shows the arrival time and burst time of each job.

TABLE XIII. Jobs in ascending order

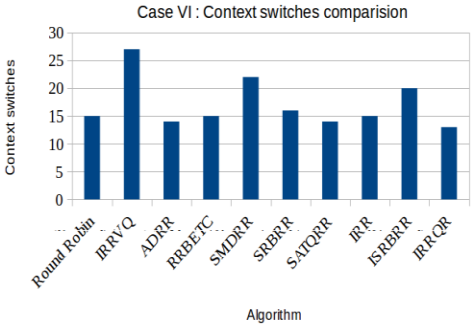
Job	P1	P2	P3	P4	P5	P6	P7
Arrival Time	2	3	0	5	17	9	12
Burst Time	31	26	3	11	85	40	76

Table 14 shows the comparison results of the algorithms

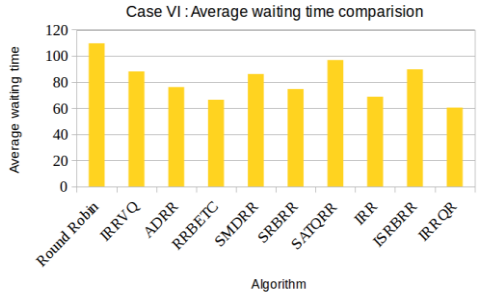
TABLE XIV. Comparison Results

Algorithm	**CS**	**AWT**	**ATT**
Round Robin	15	109.6	148.4
IRRVQ	27	88.1	127
ADRR	14	76.1	115
RRBETC	15	66.4	105.3
SMDRR	22	86.1	125
SRBRR	16	74.6	113.4
SATQRR	14	96.8	135.7
IRR	15	68.7	107.6
ISRBRR	20	89.7	128.6
IRRQR	13	60.4	99.2

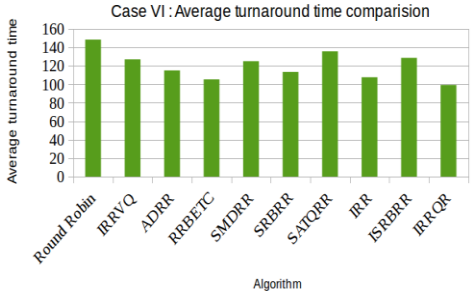
Figure 6 shows the comparison of selected algorithms with the proposed algorithm. Algorithms are shown along the x-axis while the y-axis presents no of context switches, average waiting time, and average turnaround time in figures 6a, 6b, and 6c respectively. The figure presents that our IRRQR algorithm gives the least no of CS, AWT, and ATT as compared to all existing algorithms while RRBETC and IRR give competitive results. Our IRRQR gives 15.3%, 9.9%, and 6.1% least CS, AWT, and ATT as compared to RRBETC.



(a) Context Switches



(b) Average Waiting Time



(c) Average Turnaround Time

Fig V: Comparison Results with random burst times for non-zero arrival time

5. Conclusion and Future Work

RR algorithms are widely used in real-time operating systems but their performance badly suffers from the wrong selection of time quantum. An optimal time quantum may decrease turnaround time, waiting time, and the numberof context switches. In this paper, we propose an improved quartile and randomization based dynamic round-robin scheduling algorithm (IRRQR) for optimal time quantum computation. IRRQR combines the concept of median quartile, upper quartile, and randomness in dynamic time quantum selection by providing a fair share of CPU time to all processes ensuring efficient allocation of the CPU time to different processes, preventing long-running processes from monopolizing the CPU. The quantum time is randomly selected based on the median of the burst time of all jobs and the upper quartile. Then, each job gets its time quantum. As many of the jobs complete their burst, therefore hence results in a decreasing number of context switches. The dynamic quantum is computed after one pass resulting in low complexity. Simulation results showed that our IRRQR algorithm results in reducing number of context switches up to 25%, average waiting time up to 13.7%, and average turnaround time by 8.4% as compared to existing algorithms.

In the future, we intend to use a meta-heuristic BAT algorithm with our proposed technique for large-scale scheduling problems in the cloud for optimal resource scheduling.

AUTHOR CONTRIBUTION

Asif Yar executed the research, whereas Bushra Jamil and Humaira Ijaz conceived the idea and supervised the work.

DATA AVAILABILITY STATEMENT

Not applicable.

CONFLICT OF INTEREST

The Authors declare that there is no conflict of interest.

FUNDING

(No funding available)

REFERENCES

[1] K. E. Arzen, A. Cervin, J. Eker, L. Sha, "An introduction to control and scheduling co-design", In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, Vol. 5, Sydney, NSW, Australia, 2000, pp. 4865-4870.

[2] A. Noon A. Kalakech, S. Kadry, "A new round robin based scheduling algorithm for operating systems: dynamic quantum using the mean average, *International Journal of Computer Science Issues*, Vol. 8, No. 1,2011, pp. 224-229.

[3] S. K. Panda, D. Dash, J. K. Rout, "A Grouped Based Time Quantum Round Robin Algorithm Using Min-Max Spread Measure", *International Journal of Computer Applications*, Vol. 64, No. 10, 2013, pp. 1-7.

[4] A. Silberschatz, P. B. Galvin, *Operating System Concepts*, 9th Edition, John Wiley & Sons, Inc., 2012.

[5] S.N. Gowda, "Statistical Approach to Determine Most Efficient Value for Time Quantum in Round Robin Scheduling." *AIRCC's International Journal of Computer Science and Information Technology* 8, 2016, pp. 33-39.

[6] M. K. Mishra, D. F. Rashid, "An Improved Round Robin CPU Scheduling Algorithm With Varying Time Quantum", *International Journal of Computer Science, Engineering and Applications*, Vol. 4, No. 4, 2014, pp. 1-8.

[7] U. Shafi, M. Shah, A. Wahid, K. Abbasi, Q. Javaid, M. Asghar, "A Novel Amended Dynamic Round Robin Scheduling Algorithm for Time shared Systems", *International Arab Journal of Information Technology*, Vol.17, No. 1, 2020, pp. 90-98.

[8] Y. Berhanu, A. Alemu, M. K. Mishra, "Dynamic Time Quantum based Round Robin CPU Scheduling Algorithm", *International Journal of Computer Applications*, Vol. 167, No. 13, 2017, pp. 48-55.

[9] M. LaxmiJeevani, T. S. P. Madhuri, Y. S. Devi, "Improvised Round Robin Scheduling Algorithm and Comparison with Existing Round Robin CPU Scheduling Algorithm", *IOSR Journal of Computer Engineering*, Vol. 20, Issue. 3, Version I, 2018, pp. 01-04.

[10] M. M. Tajwar, M. N. Pathan, L. Hussaini, "CPU

Scheduling with a Round Robin Algorithm Based on an Effective Time Slice", *Journal of Information Processing System*, Vol. 13, No. 4, 2017, pp. 941-950.

[11] S. K. Bhoi, S. K. Panda, D. Tarai, "Enhancing CPU performance using Subcontrary Mean Dynamic Round Robin (SMDRR) Scheduling Algorithm", *Journal of Global Research in Computer Science*, Vol. 2, No. 12,2011, pp. 17-21.

[12] A. R. Dash, S. KK. Sahu, S. K. Samantra, "An Optimized Round Robin CPU Scheduling Algorithm with Dynamic time Quantum", *International Journal of Computer Science, Engineering and Information Technology*, Vol. 5, No. 1, 2015, pp. 7-26.

[13] A. E. A. Agha, S. J. Jassbi, "A New Method to Improve Round Robin Scheduling Algorithm with QuantumTime Based on Harmonic-Arithmetic Mean (HARM)", *International Journal of Information Technology and Computer Science*, Vol. 5, No. 7, 2013, pp. 56-62.

[14] P. R. Mohanty, P. H. S. Behera, K. Patwari, M. R. Das, M. Dash, Sudhashree, "Design and Performance Evaluation of a New Proposed Shortest Remaining Burst Round Robin (SRBRR) Scheduling Algorithm", *In Proceedings of International Symposium on Computer Engineering & Technology (ISCET)*, Vol. 17, 2010, pp. 126-137.

[15] R.J. Matarneh, "Self-Adjustment Time Quantum in Round Robin algorithm Depending on Burst Time of Now Running Processes", *American Journal of Applied Sciences*, Vol. 6, No. 10, 2009, pp. 1831-1837.

[16] D. Nayak, S. K. Malla, D. Debadarshini, "Improved Round Robin Scheduling using Dynamic Time Quantum", *International Journal of Computer Applications*, Vol. 38, No. 5, 2012, pp. 34-38.

[17] P. S. Varma, "Improved Shortest Remaining Burst Round Robin (ISRBRR) Using RMS as its time quantum", *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, Vol. 1, No.8, 2012, pp. 60-64.

[18] D. Khokhar, M. A. Kaushik, "Median Based Round Robin CPU Scheduling Algorithm", *International Journal of Computer Science and Information Technology Research*, Vol. 5, No. 2, 2017, pp. 198-203.

[19] C. Zhang, P. Luo, Y. Zhao, J. Ren, "An Efficient Round Robin Task Scheduling Algorithm Based on a Dynamic Quantum Time", *International Journal of Circuits, Systems and Signal Processing*, Vol. 13, 2019, pp. 197-204.

[20] S. Z. Iqbal, H. Gull, S. Saeed, M. Saqib, M. Alqahtani, Y.A. Bamarouf, G. Krishna, and M.I. Aldossary, "Relative Time Quantum-based Enhancements in Round Robin Scheduling", *Computer Systems Science & Engineering*, Vol. 41 No. 2, 2022, pp. 461-477.

[21] S. Najafi, S. Noferesti, "Determining dynamic time quantum in round-robin scheduling algorithm using machine learning*", Soft Computing Journal*, Vol. 10 No. 2, 2022, pp.32-43.

[22] K. Vayadande, A. Bodhankar, A. Mahajan, D. Prasad, R. Dhakalkar, S. Mahajan. "An Improved Way to Implement Round Robin Scheduling Algorithm" *International Conference on Computer Vision, High-Performance Computing, Smart Devices, and Networks*, 2022, pp. 403-414.

[23] C. Sharma, S. Sharma, S. Kautish, S. A. Alsallami, E.M. Khalil, A.W. Mohamed. "A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time" *Alexandria Engineering Journal*, vol. 61, no. 12. pp. 10527-10538.

[24] N. M. Najm, "New hybrid priority scheduling algorithm based on a round Robin with dynamic time quantum." *In AIP Conference Proceedings*, vol. 2787, no. 1. AIP Publishing, 2023.