Dynamic Round Robin with Adaptive Time Quanta (DRRATQ): A Proposed Algorithm

SHABBIR HASSAN
Department of Computer Science
Aligarh Muslim University
Aligarh, 202002, India
E-mail: shassan.cs@amu.ac.in

The operating system (OS) acts as a resource manager whose responsibility is to manage the resources of a computer system. Among all resources, the CPU is one of the most crucial resources that manage the processes. Process management is achieved by a specific type of algorithm called CPU scheduling algorithm. CPU scheduling is a vital task of the OS and the whole performance of the system depends on the CPU scheduling criteria such as reducing waiting time, turnaround time, response time, and number of context switches, while enhancing the CPU utilization. Several well-known CPU scheduling algorithms like First-Come-First-Served (FCFS), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Highest Response Ratio Next (HRRN), Round Robin (RR), and Priority Scheduling algorithms are coming into the picture. In time-shared environment, RR CPU scheduling is preferred, but the system's performance depends on choosing the most appropriate time quantum. By fusing the advantages of RR with features of SJF, this paper provides an intuitive approach to enhance the conventional RR CPU scheduling algorithm with adaptive time quanta that intends to improve system performance over the improver version like IRRVQ, MRR, Tajwar et al. and Fiad et al. The work offers experimental evidence that the proposed algorithm Dynamic Round Robin with Adaptive Time Quanta (DRRATQ) performs better than the conventional RR and other existing work, by decreasing the waiting time, turnaround time, response time, and number of context switches. Implementing this algorithm to a time-sharing or distributed environment will undoubtedly improve system performance and help avoid issues like thrashing, incorporate aging, CPU affinity, and starvation. Since the proposed scheduling is work-conserving in nature, it can be used for statistical multiplexing and best-effort packet switching in a network packet scheduling environment.

Keywords: Adaptive scheduling, Jarque-Bera, thrashing, kurtosis, CPU burst, quanta, context switching, kernel density.

1. INTRODUCTION

In multiprogramming computing systems, inefficient CPU utilization can lower system performance. The time a process spends in the CPU is known as its burst, a CPU burst comprises subsequent slots of CPU burst and I/O burst. In order to effectively utilize CPU bursts, processes are stored in the memory (RAM). By switching the processes (resume and active process) between CPU and ready queue, the CPU utilization (throughput) can be optimized [1]. The main goal of a CPU scheduling algorithm is to reduce the *waiting time, turnaround time, response time, and context switches* ***(metrics category 1)*** while increasing the '*throughput and system performance* ***(metrics category 2)*** [2]. This paper presents a novel CPU scheduling algorithm called ***Dynamic Round Robin with Adaptive Time Quanta (DRRATQ)*** that achieved a better tradeoff between performance metrics listed in category 1 and category 2 over the existing algorithms IRRVQ, MRR, Tajwar *et al.* and Fiad *et al.*

1.1 Pre-requisite

When a process is submitted to a system and is waiting for a CPU burst, it is added to a queue called the ready queue. To make the most effective use of the CPU burst, it should be kept busy as much as possible. A waiting process is chosen from the ready queue whenever the CPU is idle, and then it is assigned to the CPU for execution. The time a process spends in the CPU is known as its burst time, while the time it joins the ready queue is known as its arrival time. Turnaround time refers to the amount of time a process takes to complete its execution from submission to competition of the task. The amount of time a process waits in the ready queue is referred to as its waiting time [1]. CPU scheduling algorithms work effectively to schedule processes from the ready queue to decrease the performance matrices listed in Category 1 [1].

1.2 Conventional CPU Scheduling Algorithms

The OS uses a variety of CPU scheduling methods to choose a process/task from the ready queue. This section explains some conventional CPU scheduling algorithms**.** (i) **FCFS**: The term FCFS stands for *'First Come First Serve'*, in this algorithm, processes are carried out in the order that they have arrived in the ready queue. (ii) **SJF**: The term SJF stands for *'Shortest Job First'*, the process/job with the shortest estimated execution time is executed first. (iii) **SRTF**: The term SRTF stands for *'Shortest Remaining Time First'*, it selects the process with the smallest remaining execution time to execute next. This algorithm is a preemptive version of SJF and aims to minimize the average turnaround time by giving preference to shorter processes. (iv) **HRRN**: The term HRRN stands for *'Highest Response Ratio Next'* is one of the most optimal scheduling algorithms. This is a non-preemptive algorithm in which, the scheduling is done on the basis of an extra parameter called Response Ratio. (v) **RR**: Each process is given a fixed time slice to execute, and then it is preempted to allow another process to execute. (vi) **Priority Scheduling**: Processes are assigned a priority, and higher-priority (lower the priority value) processes are executed first. (vii) **Multilevel Queue**: Processes are assigned to different priority queues, and each queue has its own scheduling algorithm. (viii) **Multilevel Feedback Queue**: Processes are assigned to a priority queue based on their current state, and they can move between queues based on their CPU burst time and other factors [3].

1.3 The problem associated with RR

Round-robin scheduling is a popular CPU scheduling algorithm that offers several benefits; however some potential problems are associated with it, they are:

(A) High Overhead

RR scheduling offers high overhead because the CPU must constantly switch between processes. This overhead can be particularly high if the time quanta is too short, which can lead to decreased performance due to thrashing.

(B) Poor Performance with Long-running Processes

RR scheduling may not perform well when there are long-running processes in the system. If a process has a long time quanta, it can hold on to the CPU for a long time, which leads to other processes waiting prolonged.

(C) Inefficient for I/O Bound Processes

RR scheduling may not be efficient for I/O bound processes. These processes typically spend most of their time waiting for I/O operations to complete and do not need to use the CPU for extended periods. In this case, RR may result in unnecessary context switches and overhead.

(D) Selection of Time Quanta

Selection of an appropriate time quantum can be challenging. If the time quantum is too short, it can result in high overhead (thrashing), while if the time quantum is too long, it can lead to decreased responsiveness and fairness (FCFS).

In general, we can say that, besides offering several benefits, such as fairness, efficiency, responsiveness, and preemption, RR also suffers from some potential problems, like high overhead (thrashing), poor performance with long-running processes (FCFS), inefficiency for I/O bound processes, and the need to select an appropriate time quantum [4]. To overcome these limitations, various modified and hybrid CPU scheduling algorithms have been proposed, they are listed in section 2.

2. RELATED WORK

Round Robin has been the most popular option for CPU scheduling because it is one of the starvation-free and easiest algorithms to implement. Nonetheless, a lot of academics have suggested, modified, and enhanced this algorithm to its fundamental functionality. As a result, numerous theories have been put forth and research has been done. The following is a description of the most important ones.

In 2009, Matarneh, R. J. presented a paper titled ***'Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes: SARR'*** [5]. The median of the processes' burst times is computed and utilized as the time quantum in SARR for each cycle. No comparison with other algorithms is provided. Similar algorithms DQRRR are also used by H.S. Behera et al. [6]. However, while they're doing it, they rearrange the procedure once more. The procedure with the shortest burst time is chosen first, followed by the one with the longest burst time, the second-lowest burst time, and so on as explained by [7].

In 2010, Behera et al. proposed a new variant of the RR scheduling algorithm, known as the '***Dynamic Quantum with Readjusted Round Robin (DQRRR)***' algorithm, which drastically decreases context switching. However, the author compares their algorithm only with the conventional RR [8]. In general, the DQRRR is to be considered an improvement over SARR [5].

In 2010, Yadav, R. K., et al. published a paper entitled 'An improved round robin scheduling algorithm for CPU scheduling' that employs the SJF algorithm to determine the next process after allocating the CPU to all processes in the first round [9].

In 2010, Ajit Singh et al. [10] mixed the idea of SJF and doubled the time quanta after every cycle and presented the title '*An Optimized Round Robin Scheduling Algorithm for CPU Scheduling*'.

In 2011, Noon et al. presented a novel algorithm known as AN, predicated upon a novel methodology termed the '*Dynamic Quantum Using the Mean Average (DQMA)*'. The premise of this method is that the burst time of the set of waiting processes in the ready queue determines how the operating system should modify the time quantum. Simulations and testing show that this technique improves the performance of RR while solving the fixed-time quantum problem [11]. Similar to this, [12] proposes an enhanced RR technique that works well in time-shared systems. On the other hand, it requires more context switching, longer waiting time, and faster response time, hence it is not appropriate for lenient real-time systems.

In 2011, Mohanty et al. proposed the '*Priority Based Dynamic Round Robin (PBDRR) Algorithm with Intelligent Time Slice for Soft Real-Time Systems* ' The algorithm determines an intelligent time slice for every process and modifies it after each execution round. The dynamic time quantum notion is used to build the suggested scheduling method [13]. The suggested algorithm performs better than the algorithm proposed [14].

In 2013, Goel et al. presented a paper '*Simulation of an Optimum Multilevel Dynamic Round Robin Scheduling Algorithm*' and developed a novel approach (a software or a simulator) called *Optimum Multilevel Dynamic Round Robin Scheduling (OMDRRS)*. It takes two random numbers, k (as time quantum) and f which calculates intelligent time slice and warps after every round of execution. If the process's remaining burst time is less than q/f while it is being executed, it will continue; if not, it will halt and move to the end of the ready queue [15].

In 2013, Ramakrishna and Rao presented a novel design '*EFFICIENT ROUND ROBIN CPU SCHEDULING ALGORITHM FOR OPERATING SYSTEMS*' where the conventional RR is optimized by adding the concept of priority scheduling approach that reduced the degree of starvation up to a minimal extent. However, the author compared their work only with the conventional RR [16].

In 2014, Mishra, M. K., & Rashid, F. presented a paper '***An improved round robin CPU scheduling algorithm with varying time quantum (IRRVQ)***' where an algorithm is suggested that makes use of two queues, ARRIVE and REQUEST; in comparison to [9], this algorithm performs better [17].

In 2014, Lee et al. published the title '*Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling*' with an adaptive round-robin approach with an emphasis on determining the optimal time quantum. The time quantum is computed after the processes are initially sorted according to their burst times, with the shorter processes at the front of the ready queue. The time quantum is equal to the average burst time of all the processes in the ready queue if the number of processes is even [18].

In 2014, Verma et al. proposed a paper '*A Round Robin Algorithm using Mode Dispersion for Effective Measure*' that subtracts the minimum burst time from the highest burst time to determine the time quantum for each cycle [19].

In 2014, Mishra, M. K., & Rashid, F. presented an '***Improved Round-Robin CPU Scheduling Algorithm with Varying Time Quantum (IRRVQ)***' that combined SJF with RR such that the burst time of the shortest process was selected as the new time quantum in each cycle and improve the conventional RR [17]. The author doesn't provide any comparison matrix of IRRVQ with other algorithms except for the conventional RR.

In 2017, Singh and Agrawal presented a paper entitled '***Modified Round Robin Algorithm (MRR)***' that proposed a median-based CPU scheduling algorithm. It combines the RR, priority, and SJF to take into consideration all three parameters such as burst time, arrival time, and priority of the process, and developed a hybrid model of them called MRR [20]. The MRR is compared against conventional RR, New Improved Round Robin (NIRR) [21] and Subcontrary Mean Dynamic Round Robin (SMDRR) (presented under title 'ENHANCING CPU PERFORMANCE USING SUBCONTRARY MEAN DYNAMIC ROUND ROBIN (SMDRR) SCHEDULING ALGORITHM' by Bhoi et al. [22]), and is found better in performance metrics listed in category 1 and 2 as well.

In 2017, Tajwar et al. presented a paper entitled '***CPU Scheduling with a Round Robin Algorithm Based on an Effective Time Slice***'. The proposed algorithm is found effective due to dynamic quanta allocation to the process at each round of the execution. The algorithms were compared with conventional RR and other established algorithms such as IRRVQ, DQRRR, SARR, and MRR and found better in performance metrics listed in categories 1 and 2 [23].

In 2019, Li et al. presented a *Two-Phase Optimized Round-Robin Algorithm (TPORRA)*. Similar to the conventional round-robin algorithm, the first phase handles processes that must be carried out in sequence; each process is given a runtime of a single time slice. The second phase involves doubling the time quantum and executing the processes according to the number of burst times they have left, with shorter operations being run before longer ones. After every process is finished, the first step is repeated [24].

In 2020, Fiad et al. published a paper entitled 'Improved Version of Round Robin Scheduling Algorithm Based on Analytic Model' that presents an improved RR CPU scheduling algorithm with varying time quantum. The authors have used variable time quanta with an analytical model that mainly focuses on the order of the task/process taken from the ready queue for execution. It has eliminated the drawbacks faced by conventional RR and made some improvements over the existing work/algorithms IRRVQ, Tajwar et al., and average execution time-round-robin (AET-RR) presented in [25]. The algorithm is mainly designed for cloud computing and distributed environments [26].

In 2021, Mostafa et al. proposed a title '*ATS: A Novel Time-Sharing CPU Scheduling Algorithm Based on Features Similarities*' that presents a clustering-based novel design Adjustable Time Slice (ATS). The ATS is compared against well-known existing works like RR, DTS, VTRR, and ATRR [27].

In 2022, Sharma et al. published a paper '*Modified Round Robin CPU Scheduling: A Fuzzy Logic-Based Approach*' which presents a fuzzy logic-based approach that overcomes the limitation of conventional RR and optimizes waiting time and turnaround time. However, the author compared their work only with RR and FCFS [28].

3. PROPOSED SCHEDULING ALGORITHM DRRATQ

The proposed algorithm '*Dynamic Round Robin with Adaptive Time Quanta (DRRATQ)*' combines the features of RR and SJF with IRRVQ, Tajwar et al. and Fiad et al. to overcome the limitation faced by them [1, 17, 23, 26]. The DRRATQ CPU scheduling algorithm comes with an adaptive time quantum feature that selects/assigns the quantum to processes based on the lapsed quanta of completed (halted) tasks and the locality of quanta of forthcoming processes that are still waiting in the ready queue. The proposed method sorts all process's burst times in such a way that an optimal measure of central tendencies is found.

Initially, we arrange all processes in increasing order of their CPU burst (comprises both CPU burst and IO burst) time and set them into a ready queue. To eliminate overhead we set the simulation seed with the smallest burst time of the process. In order to reduce the extreme CPU utilization of the last process (as faced in IRRVQ, SARR, and Tajwar et al.), the algorithm combines the burst time of the processes with noisy bursts to get a desirable CPU quantum. The term AWT, AT, ATT, and NCS indicates average waiting time, arrival time, average turnaround time, and the number of context switch respectively. It is noted that a process burst requirement may lie in either outlier (left-hand side or right-hand side) and hence they are solved by a proper amalgamation of DQRRR and SARR, whereas in the rest of the case, the required quanta are calculated as follows:

To calculate quanta, a mathematical model (polynomial fitting model based on dataset CPU quanta and CPU burst named as qb_set) is designed that will predict a CPU quanta by combining the quanta generation method of the ‘basis algorithms: IRRVQ, MRR, Tajwar et al. and Fiad et al.’. It is noted that among these basis algorithms, Fiad et al. achieved significant AWT and ATT while the minimum NCS was achieved by Tajwar et.al. (refer to Table 6 and Table 8) of sections 4.2.1 and 4.2.2 respectively. The CPU quanta generated by these basis algorithms serve as a training set for DRRATQ so that the accuracy and transparency of the proposed algorithm are justified. The DRRATQ is implemented in ‘R’ and is based on a non-linear to polynomial regression. It uses the core statistical measures such as ‘mean, standard deviation, chi-squared Test, adjusted R-squared, polynomial fitting, and kernel density function (KDF)’ to decide the best quanta. The proposed scheduling is dynamic in nature because it decides the next CPU burst in reference to the quanta that have been used previously (like a combinational logic circuit), and is also adaptive in nature because, at each pass, it uses ‘d’(5, 5, 4, 3, 2) degree of freedom (DF) (because at each pass, one process gets completed and previous quanta is used as adaptive coefficient to calculate next quanta) to predict the next CPU quanta by using Jarque-Bera (JB) Test. The JB test is used to determine the Skewness (symmetry of the distribution of quanta density and noisy burst) and Kurtosis (sharpness of the curve found in the experiment) of the remaining CPU bursts w.r.t. the previously used quanta. The scatter plot generated by qb_set (quanta vs. burst) is shown in Fig. 1, while the quanta density vs. noisy burst is shown in Fig. 2. The formula to calculate the JB coefficient is given in equation 1.

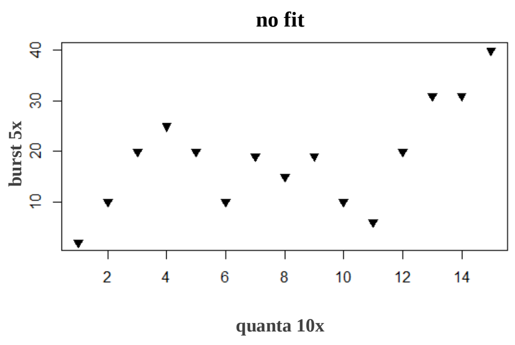


Fig. 1: Scatter plot of generated by qb_set

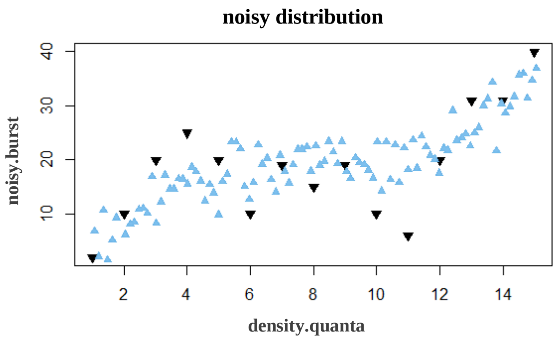


Fig. 2: quanta density vs. noisy.burst

$$\boldsymbol{JB} = \boldsymbol{n}\left[\frac{\boldsymbol{k_3}^2}{\mathbf{6}} + \frac{\boldsymbol{k_4}^2}{\mathbf{24}}\right], \textit{for} \qquad (1)$$

$$k_3 = \frac{\sum_{i=1}^{n}\left(b_i - \bar{b}\right)^3}{ns^3}$$

$$k_4 = \frac{\sum_{i=1}^{n}\left(b_i - \bar{b}\right)^4}{ns^4} - 3$$

Where $\boldsymbol{n}$ is the sample size (no. of process), $\boldsymbol{k_3}$ represents the Skewness, $\boldsymbol{k_4}$ represents the Kurtosis of each CPU burst in each $\boldsymbol{s}$ standard deviation and $\boldsymbol{b}$ represents the CPU burst requirement of the processes. Now we are going to design a data frame consisting order pair of quanta and burst.

qb_set ← *data.frame(quanta = 1:15, burst = c(31, 10, 20, 6, 20, 10, 40, 15, 19, 10, 25, 20, 31, 2, 19))*

quanta (q)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
burst (b)	2	10	20	25	20	10	19	15	19	10	6	20	31	31	40

This can be visualized as follows (refer to Fig. 1).

plot(qb_set$quanta, qb_set$burst, pch = 25, xlab = 'quanta 10x', ylab = 'burst 5x', main = "no fit")
set.seed(b[P0])

In order to add non-linearity, some noise is generated (with the same mean and sd of the data frame in qb_set) and added to the actual CPU burst to get a set of quanta density.

```
noise ← rnorm(length(quanta), mean = 69.4, sd = 25.72)
noisy.burst ← burst + noise
```

Plot of the noisy burst (refer to Fig. 2).

```
plot(noisy.quanta, noisy.burst, col = 'skyblue2', xlab = 'density.quanta', ylab = 'noisy.burst')
```

Fig. 2 represent the plot of our simulated observed data where so many noisy burst is mixed with the actual CPU burst and rnorm quanta density is formed. Now we are going to fit a polynomial by using regression models to these data points in reference to the qb_set, and exhibit each model's curve in a single plot with up to degree-6 polynomial regression models.

```
model[1] ← lm(burst~quanta, data = qb_set)
model[1]
```

Output:

```
Coefficients:
(Intercept)        quanta
    7.533           1.375
```

```
model[2] ← lm(burst~poly(quanta, 2, raw = TRUE), data = qb_set)
model[2]
```

Output:

```
Call:
lm(formula = burst~poly(quanta, 2, raw = TRUE), data = qb_set)

Coefficients:
        (Intercept)  poly(quanta, 2, raw = TRUE)1  poly(quanta, 2, raw = TRUE)2
          15.1341                 -1.3076                   0.1677
```

Until now, the intermediate equation of the fitting curve, which will best fit the CPU burst with a desirable quantum, is as follows:

$$\boldsymbol{y = \lambda x^2 + \mu x + c}$$

for all,

$\boldsymbol{\lambda}$	$\boldsymbol{\mu}$	$\boldsymbol{c}$
0.1677	-1.3067	15.1341

```
for(d in 3:6){
  model[d] ← lm(burst~poly(quanta, d, raw = TRUE), data = qb_set)
}
```

Until now, we have created six models that possibly fit the CPU burst with a desirable quantum. Fig. 3(A), 3(B), 3(C), 3(D), 3(E) and 3(F) represents the graphical significance of each model.

```
plot(qb_set$quanta, qb_set$burst, pch = 25, xlab = 'quanta 10x', ylab = 'burst 5x', main = "possible fits")

Q ← seq(1, 15, length = 15)

rainbow ← c("#FF0000FF", "#FFBF00", "#FF00FFFF", "#00FF00FF", "#00FFFFFF", "#0000FFFF")
```

The predict( ) function is used to forecast the values based on the previous data behaviors and thus by fitting that data to the model.

```
for(i in 1:6){
  lines(Q, predict(model[i], data.frame(quanta = Q)), col= rainbow[i], lty = 1, lwd = 1.5)
}
```

fit: 30.37% **fit: 34.14%**

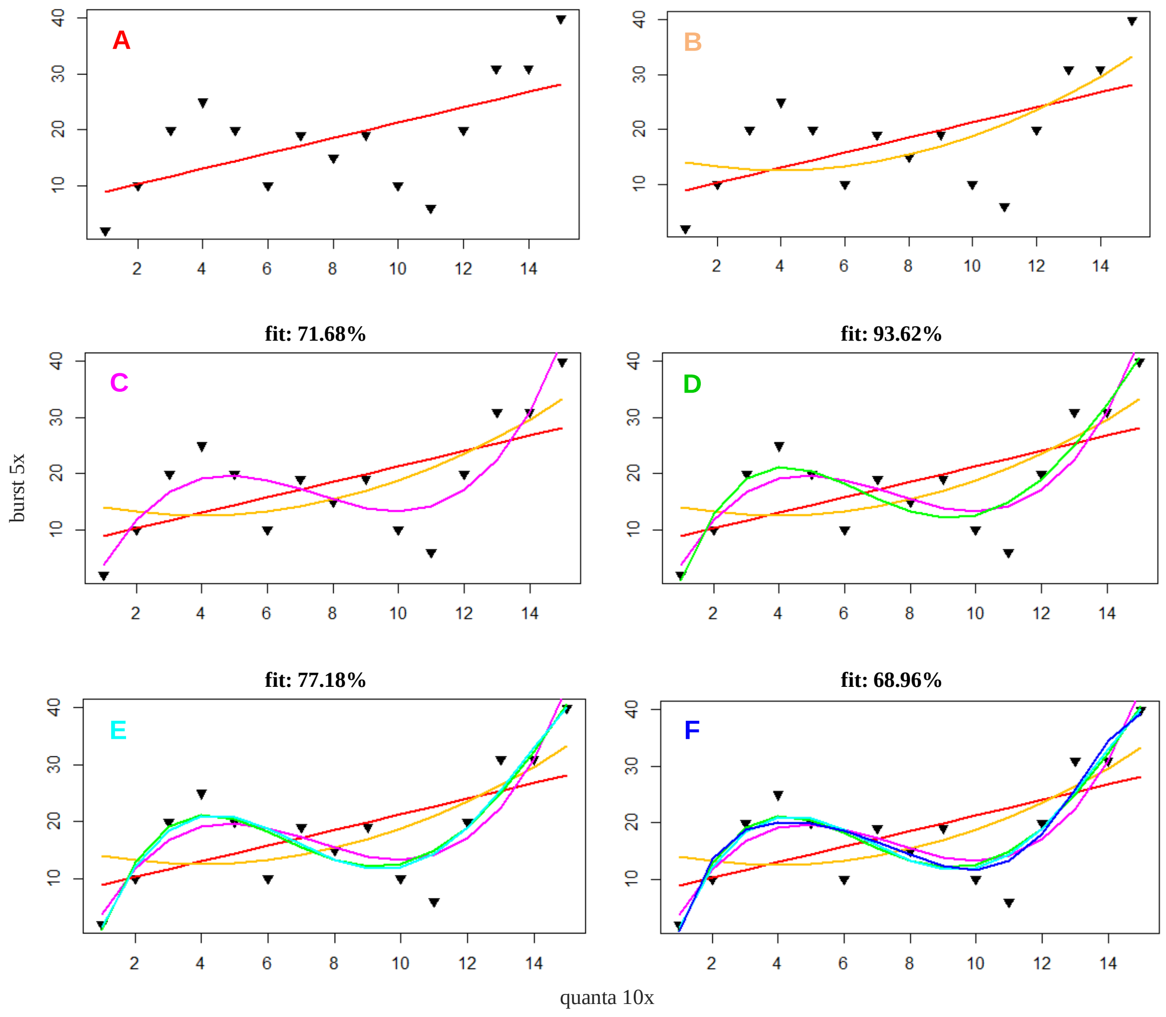


Fig. 3 (A-F): Coefficient of goodness-of-fit achieved by polynomial regression models

Now, we are going to use the adjusted R-squared regression on each model to determine which curve best matches a CPU quanta with each process's burst. This score indicates how much of the variation in the response variable (quanta) is achieved by the model's predicte (process's burst). The summary( ) method with adj properties is used to calculate adjusted R-squared coefficients:

```
while(i in 1:6){
    summary(model[i])$adj.r.squared
}
```

Table 1: Coefficients of adjusted R^2

models	**adj(R^2)**	**remarks**
model[1]	30.78%	worst
model[2]	34.14%	-
model[3]	71.68%	-
model[4]	**93.62%**	**best**
model[5]	77.18%	optimal
model[6]	68.96%	-

As we can see in Table 1, the fourth-degree polynomial achieved the highest adjusted R-squared value which is 0.9362472 (93.62472%, refer to Fig. 3(D)). Finally, by creating a mental image of this model (model[4] refer to Fig. 3(D)), equation 2 represents the required polynomial with Estimated Std. Errors (α, β, γ, and δ) with intercepts 'c' which further rectified by the JB test. The scatterplot of fourth-degree polynomial is depicted in Fig. 4. While Fig. 5 represents the KDF of the model.

*plot(**qb_set**$quanta, **qb_set**$burst, pch = 25, xlab = 'quanta 10x', ylab = 'burst 5x', main = "**best fit**")*

Q ← seq(1, 15, length = 15)

*lines(**Q**, **predict**(model[4], data.frame(quanta = Q)), col = rainbow[4], lty = 1, lwd = 2)*

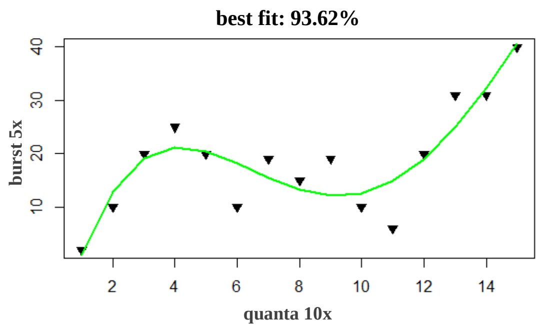


Fig. 4: Scatterplot of fourth-degree polynomial

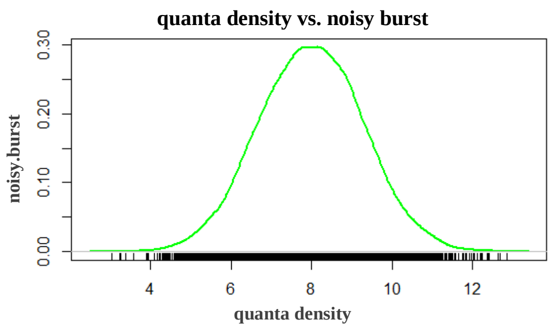
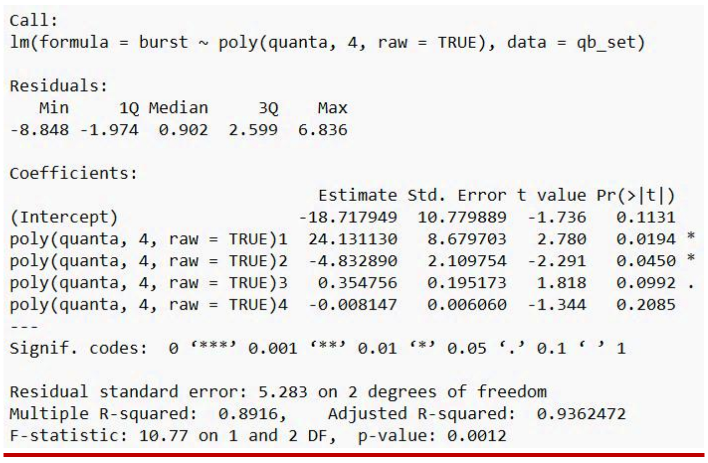


Fig. 5: Kernel density function of model[4]

The summary( ) method is used to get the coefficients of the polynomial of the line of regression.

summary(model[4])


```
Call:
lm(formula = burst ~ poly(quanta, 4, raw = TRUE), data = qb_set)

Residuals:
   Min      1Q Median      3Q     Max
-8.848 -1.974  0.902  2.599  6.836

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                 -18.717949  10.779889  -1.736   0.1131
poly(quanta, 4, raw = TRUE)1  24.131130   8.679703   2.780   0.0194 *
poly(quanta, 4, raw = TRUE)2  -4.832890   2.109754  -2.291   0.0450 *
poly(quanta, 4, raw = TRUE)3   0.354756   0.195173   1.818   0.0992 .
poly(quanta, 4, raw = TRUE)4  -0.008147   0.006060  -1.344   0.2085
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.283 on 2 degrees of freedom
Multiple R-squared:  0.8916,    Adjusted R-squared:  0.9362472
F-statistic: 10.77 on 1 and 2 DF,  p-value: 0.0012
```

The equation of the curve is as follows (y as quanta and x as burst):

$$y = \alpha x^4 + \beta x^3 + \gamma x^2 + \delta x + c \quad (2)$$

for all:

α	β	γ	δ	c
-0.008147	0.354756	-4.832890	24.131130	-18.717949

We can use this equation to predict the value of the response variable y (as quanta) based on the predicate x (as bursts) of the model[4]. Table 2 represents the response set of the model[4] based on the new predicator (not included in qb_set) that best matches the process burst requirements.

quanta = predict(model[4], data.frame(burst = 37))

Table 2: Response set of model[4] with new predictors

burst	1	4	13	25	40	102
adj(R^2)	0.926906	-21.0991	17.62909	-75.2701	14.25501	39.2801
Q	**2**	**4**	**17**	**25**	**45**	**110**

3.2 The algorithm for quanta computation of DRRATQ is given below:

Algorithm: DRRATQ

```
begin:
define a vector b_queue ← null
L:
while(i in 'n')
    If: pass == 1
        set: df ← n
        define burst vector 'b' for processes $P_0, P_1, P_2, \ldots, P_{n-1}$ with burst requirement $b_0, b_1, b_2, \ldots, b_{n-1}$, respectively
        b_queue[ i ] ← b[ $P_i$ ]
        sort (b_queue, decreasing = FALSE)
    else
        set: df ← n – i + 2
        b_queue[ j ] ← $\boldsymbol{Q}_{i-1}$ ∀ i ≥ 2 such that j = i - 1
        while(j < n) { b_queue[ ++j ] ← b_queue[ i++ ]  }
    end if
    $\mu$ ← mean(b_queue[ k ]$_{k = 0, 1, 2, \ldots, n-1}$)
    $\sigma$ ← sd(b_queue[ k ]$_{k = 0, 1, 2, \ldots, n-1}$)
    compute: JB with $\mu$ and $\sigma$, compare it with $\chi^2$ against df
    compute: $\emptyset$ with critical value 'c' and $\mu$% against the computed JB
    go through a hypothesis testing: $H_0$ or $H_1$
    if: $H_0$
        $Q \leftarrow b[P_i] + \lceil \emptyset - \lambda \rceil$
    else
        $Q \leftarrow b[P_i]$
    end if
    update: i
    Allocate all processes to CPU with generate quanta Q as per the rule*, go to L;
end
```

At each pass, processes are allocated to the CPU for quanta **Q, and then (after this pass/round) available processes are then get sorted and the immediate used quanta is placed at the position of exhausted CPU burst of the process (refer to table 5) for calculation of next quanta by using equation 3.*

3.2 Simulation

The paper presents the experimental result with five processes taken in two different cases, the case i: deals with zero AT while the case ii deal with discrete non-zero AT. In both cases, set of process burst is chosen differently so that cases explained in [17,23, 26] can be covered. The 'R-4.3.2' programming language is used to implement the algorithm and related statistical measures is calculated in RStudio. Graphical User Interface (GUI) based CPU Scheduling Simulator (https://github.com/KhaledAshrafH/CPU-Scheduling-Simulator.git) is used to run the program and simulate the experiment.

4. EXPERIMENT AND ANALYSIS

4.1 Postulate

The evaluation of performance assumes that in a single processor environment, all processes have the same priority. Before their execution, the burst time and number of processes are known. The overhead involved in switching between processes (context switching) and arranging them in the ready queue has been deemed negligible. All processes are bound to the CPU, and none of them are related to input/output. The time quantum is measured in milliseconds and have created two hypotheses H_0 and H_1 such that:

***H*₀:** *The required quanta* $\boldsymbol{Q}$ *for this pass must be greater than the CPU burst requirement* $\boldsymbol{P_i}$ *iff:* $\boldsymbol{adj(R^2)} \geq \boldsymbol{0}$

***H*₁:** *The required quanta* $\boldsymbol{Q}$ *for this pass will be equal to the CPU burst requirement of* $\boldsymbol{P_i}$ *iff:* $\boldsymbol{adj(R^2)} < \boldsymbol{0}$

Based on the adjusted R-Squared value, equation 3 calculates the required quanta $\boldsymbol{Q}$ and validates the hypothesis.

$$Q = b[P_i] + \lceil \emptyset - \lambda \rceil \tag{3}$$

$$\emptyset = \frac{JB \times C}{\psi}$$

Where $\boldsymbol{Q}$ is the required quanta, $\boldsymbol{b}$ is the CPU burst requirement of the process $\boldsymbol{p_i}$, $\boldsymbol{JB}$ represents the statistic of the Jarque-Bera Test, $\boldsymbol{C}$ is the critical value of the $\boldsymbol{\chi^2}$ distribution is taken from Rholf and Sokal Table (refer to Table 4), $\boldsymbol{\psi}$ is the mean of the square of critical values (u and v), each associated with a level of significance (α) against the critical value $\boldsymbol{C}$, $\boldsymbol{\lambda}$ is the p-value adjuster.

$$\boldsymbol{\psi} = \mu(u^2, v^2) \tag{4}$$

accepted = H₀ if $\emptyset$ ***is > 0 else H₁*** (5)

$\boldsymbol{Q} \leftarrow b[P_i] + \lceil \emptyset - \lambda \rceil$ ***if: accepted***

else

$\boldsymbol{Q} \leftarrow b[P_i]$

1.2 Experiments Performed

The proposed scheduling DRRDTQ is simulated in two different scenarios. In both the scenarios, CPU burst requirements are randomly ordered. In the first scenario, the process arrival time is assumed to be 0. While in the second scenario, the process arrival time is assumed to be non-zero. Since the CPU burst time in increasing or decreasing order produces the same results, hence we did not consider this case in the experiment. The procedure to generate quanta and process allocation is as follows:

1.2.1 CASE 1: Pi with zero AT

All five processes P0, P1, P2, P3, and P4 are taken with CPU burst requirements 60, 55, 102, 40, and 90 respectively in the sequence of execution as P3, P1, P0, P4 and P2 with burst mean $\boldsymbol{\mu(Pi)}$ as 69.4 and $\boldsymbol{\sigma(Pi)}$ as 25.72 in five DF (because pass == 1) as shown in Table 3.

Table 3: Snapshot of the scenario with zero arrival time

Process	Arrival Time	Burst Time
P_0	0	60
P_1	0	55
P_2	0	102
P_3	0	40
P_4	0	90

In the first pass, the computed JB statistic is 4.12 (see equation 1) which then compared to $\boldsymbol{\chi^2}$ distribution against DF, at alpha-level 0.05, the critical value ($\boldsymbol{C}$) is 11.070 (see Table 4). Since the JB statistic fall between 0.9 and 0.5, and is not greater than the critical value 11.070, so ψ is:

Table 4: χ^2 distribution is taken from Rholf and Sokal [29]

DF \ α	0.995	0.99	0.975	0.95	0.9	0.5	0.1	0.05	0.025	0.01	0.005
1	---	---	0.001	0.004	0.016	0.455	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	1.386	4.605	**5.991**	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	2.366	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	3.357	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	4.351	9.236	**11.070**	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	5.348	10.645	12.592	14.449	16.812	18.548

ψ = mean(1.6102, 4.3512) ≈ 10.761, from equation 4

Hence $\emptyset$ will be (4.12x11.070)/ψ = 45.6084/10.761 ≈ 4.2383.

The value of $\lambda \approx 0.0012$ (see the last line of summary(model[4])), thus $\emptyset - \lambda$ is 4.2371. Since, the suggested $\emptyset > 0$, by using equation 5, hence the NULL hypothesis H0 will be accepted and the required quanta will be calculated as:

$$\boldsymbol{Q} \leftarrow b[P_i] + \lceil \emptyset - \lambda \rceil$$

Which is equivalent to 40 + $\lceil \emptyset - \lambda \rceil \approx 45$. Hence, at the first pass, processes will serve quanta 45, and after the completion of this pass (pass 1 with quanta 45), the next quantum would be 10, 11, 28, and 10 as shown in Table 5. The AWT, ATT, and NCS found in the experiment (see Table 6) indicate that the factor $\lceil \emptyset - \lambda \rceil$ guaranteed the best trade-off between performance matrics listed in categories 1 and 2.

Table 5: Snapshot of the scenario with zero arrival time

pass	Processes					μ	σ	JB	DF	Q
	P_3	P_1	P_0	P_4	P_2					
1	40	55	60	90	102	**69.4**	**25.72**	**4.12**	**5**	**45**
	-5	**10**	**15**	**45**	**57**	60.4	41.57			
2	**45**	**10**	**15**	**45**	**57**	**34.4**	**20.65**	**3.06**	**5**	**10**
	x	0	5	35	47	21.75	22.85			
3	-	**10**	5	35	47	**24.25**	**20.05**	**3.5**	**4**	**11**
	x	x	-6	**24**	**36**	18	21.63			
4	-	-	**11**	**24**	**36**	**23.66**	**12.5**	**5.04**	**3**	**28**
	x	x	x	-4	**8**	2	8.48			
5	-	-	-	**28**	**8**	**18**	**14.14**	**4.8**	**2**	**10**
	x	x	x	x	-2	-2	NA			

Table 6 represents a fair comparison of outcomes found in DRRATQ with existing works. The Stacked bar plot of outcomes mentioned in Table 6 is shown in Fig. 6.

Table 6: Comparision matrix of the proposed algorithm in case 1

#	Algorithm	Quanta (Q)	AWT	ATT	NCS
1.	RR	25	192	261.4	16
2.	SARR	40, 15, 05, 30, 12	165	234.4	14
3.	DQRRR	15, 05, 29, 03	160.3	220	10
4.	IRRVQ	35, 33, 18, 5	167	210.6	9
5.	MRR	20, 25, 15	141.6	205	13
6.	Tajwar *et al.*	28, 25, 30, 13	129	201.4	**7**
7.	Fiad *et al.*	40, 34, 16, 22	120	190	9
8.	**DRRATQ**	**45, 10, 11, 28, 10**	**97.8**	**141.3**	**8**

1.2.2 CASE 2: Pi with non-zero AT

In this case, processes join the ready queue at discrete time instant starting from 0 and the CPU burst time is assumed to be randomly ordered. All five processes P0, P1, P2, P3, and P4 are taken with CPU burst requirements 7, 25, 5, 36, and 18 respectively (as shown in Table 7) in the sequence of execution as P0, P1, P2, P3, and P4 with burst mean $\boldsymbol{\mu(Pi)}$ as 18.2 and $\boldsymbol{\sigma(Pi)}$ as 12.87 in five DF (because no process has completed yet).

Table 7: Snapshot of the scenario with non-zero arrival time

Process	Arrival Time	Burst Time
P_0	0	7
P_1	4	25
P_2	10	5
P_3	15	36
P_4	17	18

After going through the same quanta generation process mentioned in Table 5, the obtained result is shown in Table 8 and compares the outcomes of the proposed algorithm. Among all existing works, the minimum AWT and ATT are found in Fiad et al., hence, the algorithm is compared with only conventional RR and Fiad et al. while the NCS of the proposed scheduling lies between the NCS found in Tajwar et al. and Fiad et al. The outcome found in this case is represented by a stacked bar plot in Fig. 7.

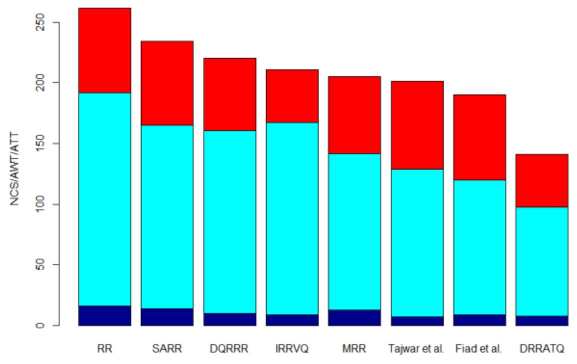


Fig. 6: Stacked bar plot of Table 6

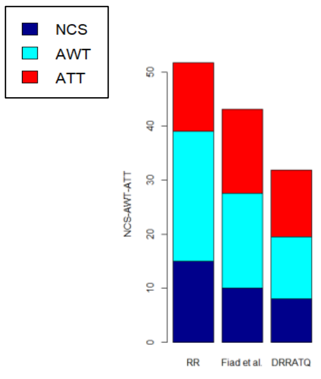


Fig. 7: Stacked bar plot of Table 8

From the above graph (refer to Fig. 6 of case 1) we can observe that, the AWT, ATT, and NCS are decreased by 49.06%, 45.94%, and 50% respectively in comparison to RR, and 18.5%, 25.63% and 11.11% respectively in comparison to Fiad et al.

From the above graph (refer to Fig. 7 of case 2) we can observe that, the AWT, ATT, and NCS are decreased by 50%, 38.29%, and 46.66% respectively in comparison to RR, and 29.19%, 25.98%, and 20% respectively in comparison to Fiad et al.

Table 8: Comparision matrix of the proposed algorithm in case 2

#	Algorithm	Quanta (Q)	AWT	ATT	NCS
1.	RR	6	39	51.7	15
2.	Fiad *et al*.	7,11	27.54	43.1	10
3.	**DRRATQ**	**6, 9, 16**	**19.5**	**31.9**	**8**

5. CONCLUSIONS

Allocating CPU to waiting processes is a crucial task of operating system, and several CPU scheduling algorithms have been developed with their respective pros and cons. The proposed scheduling DRRDTQ with adaptive time quanta outperforms the conventional RR and significantly reduces the waiting time, turnaround time, and the number of context switch in comparison to existing work like IRRVQ, MRR, Tajwar et al. and Fiad et al., and yield a better tradeoff between performance matrics listed in categories 1 and 2. Table 6 and Table 8 present a fair comparison of the outcomes found in the experiment and the simulation results (refer to Table 5) verify the validity of the experimental results. Consequently, implementing this algorithm to a distributed or time-sharing system surely enhances the system's performance which could help to prevent situations like incorporating aging, CPU affinity, starvation, and thrashing. Since the proposed scheduling is work-conserving, in network packet scheduling, this can also be implemented for best-effort packet switching and statistical multiplexing.

REFERENCES

[1] A. Silberschatz, P. B. Galvin & G. Gagne, (2005) *'Operating System Concepts'*, John Wiley and Sons Inc, pp157-167.

[2] Rami J. Matarneh, (2009) *"Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of Now Running Processes",* American J. of Applied Sciences, Vol. 6, No. 10, pp1831-1837.

[3] M Lavanya & S. Saravanan, (2013) *"Robust Quantum Based Low-power Switching Technique to Improve System Performance",* International Journal of Engineering and Technology, Vol. 5, No. 4,pp 3634-3638.

[4] Abdulrazak Abdulrahim, Salisu Aliyu, Ahmad M Mustapha & Saleh E. Abdullahi, (2014) *"An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm"*, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 4, Issue 2, pp 601-610.

[5] Matarneh, R. J. (2009). *"Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes". American Journal of Applied Sciences*, *6*(10), 1831.

[6] Behera, H. S., Mohanty, R., & Nayak, D. (2011). *"A new proposed dynamic quantum with re-adjusted round robin scheduling algorithm and its performance analysis". arXiv preprint arXiv:1103.3831*.

[7] FLEMING'S, G. A. (2012). *'The new method of adaptive CPU scheduling using fonseca and fleming's genetic algorithm'*. Journal of Theoretical and Applied Information Technology, 37(1).

[8] H.S. Behera, R. Mohanty & Debashree Nayak, (2010) "A New Proposed Dynamic Quantum with Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis", International Journal ofComputer Applications,Vol. 5,No.5, pp10-15.

[9] Yadav, R. K., Mishra, A. K., Prakash, N., & Sharma, H. (2010). *"An improved round robin scheduling algorithm for CPU scheduling". International Journal on Computer Science and Engineering*, *2*(04), 1064-1066.

[10] Singh, A., Goyal, P., & Batra, S. (2010). *"An optimized round robin scheduling algorithm for CPU scheduling". International Journal on Computer Science and Engineering*, *2*(07), 2383-2385.

[11] Abbas Noon, Ali Kalakech, and Seifedine Kadry, (2011) *"A New Round Robin based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average"*, International Journal of Computer ScienceIssues, Vol.8, Issue3, No.1, pp 224-229.

[12] Mohanty, R., Pradhan, S. C., & Behera, S. R. (2012). *"A Priority Based Dynamic Round Robin with Deadline (PBDRRD) Scheduling Algorithm for Hard Real Time Operating System". International Journal of Advanced Research in Computer Science*, *3*(3).

[13] Mohanty, R., Behera, H. S., Patwari, K., Dash, M., & Prasanna, M. L. (2011). *"Priority based dynamic round robin (PBDRR) algorithm with intelligent time slice for soft real time systems". arXiv preprint arXiv:1105.1736*.

[14] Yaashuwanth, C., & Ramesh, R. (2010). *"A new scheduling algorithm for real time system". International Journal of Computer and Electrical Engineering*, *2*(6), 1104.

[15] Goel, N., & Garg, R. B. (2013). *"Simulation of an optimum multilevel dynamic round robin scheduling algorithm". arXiv preprint arXiv:1309.3096*.

[16] Ramakrishna, M., & Rao, G. P. R. (2013). *"Efficient round robin CPU scheduling algorithm for operating systems". International Journal of Innovative Technology and Research*, *1*, 103-109.

[17] Mishra, M. K., & Rashid, F. (2014). *"An improved round robin CPU scheduling algorithm with varying time quantum", International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol*, *4*.

[18] Lee, M., Song, S., Moon, J., Kim, J., Seo, W., Cho, Y., & Ryu, S. (2014, February). *"Improving GPGPU resource utilization through alternative thread block scheduling".* In *2014 IEEE 20th international symposium on high performance computer architecture (HPCA)* (pp. 260-271). IEEE.

[19] Verma, R., Mittal, S., & Singh, V. (2014). *"A round robin algorithm using mode dispersion for effective measure". International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 166-174.

[20] Singh, M., & Agrawal, R. (2017, September). *"Modified round robin algorithm (MRR)".* In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)* (pp. 2832-2839). IEEE.

[21] Abdulrahim, A., Abdullahi, S. E., & Sahalu, J. B. (2014). *"A new improved round robin (NIRR) CPU scheduling algorithm". International Journal of Computer Applications*, *90*(4).

[22] Bhoi, S. K., Panda, S. K., & Tarai, D. (2011). *"Enhancing CPU performance using sub-contrary mean dynamic round robin (SMDRR) scheduling algorithm". Journal of Global Research in Computer Science*, *2*(12), 17-21.

[23] Tajwar, M. M., Pathan, M. N., Hussaini, L., & Abubakar, A. (2017). *"CPU Scheduling with a Round Robin Algorithm Based on an Effective Time Slice". Journal of Information processing systems*, *13*(4).

[24] Li, Z. (2019, August). *"Scheduling of Two-Phase Mixed-Criticality Systems on Memory Shared Multi-Core Platforms".* In *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)* (pp. 453-458). IEEE.

[25] Das, S. B., Mishra, S. K., & Sahu, A. K. (2019). *"An efficient average execution time-round-robin (AET-RR) scheduling algorithm". International Journal of Information Technology*, 1-14.

[26] Alaa, F. I. A. D., Zoulikha, M. M., & Hayat, B. E. N. D. O. U. K. H. A. (2020, November). *"Improved round robin scheduling algorithm with varying time quantum".* In *2020 Second International Conference on Embedded & Distributed Systems (EDiS)* (pp. 33-37). IEEE.

[27] Mostafa, S. M., Idris, S. A., & Kaur, M. (2022). *"ATS: A Novel Time-Sharing CPU Scheduling Algorithm Based on Features Similarities". Computers, Materials & Continua*, *70*(3).

[28] Sharma, R., Goel, A. K., Sharma, M. K., Dhiman, N., & Mishra, V. N. (2023, May). '*Modified Round Robin CPU Scheduling: A Fuzzy Logic-Based Approach'*. In Applications of Operational Research in Business and Industries: Proceedings of 54th Annual Conference of ORSI (pp. 367-383). Singapore: Springer Nature Singapore.

[29] S. C. Gupta and V. K. Kapoor, *'Fundamental of Mathematical Statistics'*, ISBN: 978-81-8054-528-3 (11th edition), First Published: Sep. 1970.

Shabbir Hassan (沙比尔·哈桑) received the Ph.D. Degrees in Computer Science from Aligarh Muslim University, India. He is currently working as an Assistant Professor at the Department of Computer Science, Aligarh Muslim University, India. His research interests include Computing, Stream Cipher, Cryprography and Cryptanalysis, AL and ML.

Maria Qamar (玛丽亚·卡玛尔) received the B.S. and M.S. (MCA Gold Medalist) Degrees in Computer applications from Vinoba Bhave University, India and currently pursuing Ph.D. at Radha Govind University, India. Her research interests include Data Science, System Analysis and Design, Machine Learning, Machiene Optimization and Scheduling.