ORR: Optimized Round Robin CPU Scheduling Algorithm

Conference Paper · August 2021

DOI: 10.1145/3484824.3484917

5 authors, including:

Amit Kumar Gupta
Amity University
54 PUBLICATIONS **304** CITATIONS
SEE PROFILE

Priya Mathur
Poornima Group of Colleges
78 PUBLICATIONS **453** CITATIONS
SEE PROFILE

Carlos M. Travieso
University of Las Palmas de Gran Canaria
455 PUBLICATIONS **5,976** CITATIONS
SEE PROFILE

Muskan Garg
Panjab University
55 PUBLICATIONS **474** CITATIONS
SEE PROFILE

ORR: Optimized Round Robin CPU Scheduling Algorithm

Amit Kumar Gupta†
Amity School of Eng. & Tech.,
Amity University Rajasthan,
Jaipur, Rajasthan, India-303007
dramitkumargupta1983@gmail.com

Priya Mathur
Department of Mathematics
Poornima Institute of Eng. & Tech
Jaipur, Rajasthan, India- 302022
drpriyamathur21@gmail.com

Carlos M. Travieso-Gonzalez
University of Las Palmas de Gran Canaria, Spain, Spain- 35001
carlos.travieso@ulpgc.es

Muskan Garg
Thapar Institute of Eng. & Tech
Patiala, Panjab, India- 147004
muskanphd@gmail.com

Dinesh Goyal
Department of CSE
Poornima Institute of Eng. & Tech
Jaipur, Rajasthan, India- 302022
dinesh8dg@gmail.com

ABSTRACT

The time-specific applications are assigned to Central Processing Unit (CPU) of the system and one of the most promising functions of the time-sharing operating systems is to schedule the process in such a way that it gets executed in minimal time. At present, the Round Robin Scheduling Algorithm (RRSA) is the most widely used technique in a time-sharing operating system because it gives better performance than other scheduling techniques, namely, First Come First Serve (FCFS), Shortest Job First (SJF), and Priority scheduling. The major challenge in RRSA is the static value of Time Quantum (TQ) which have plays a pivotal to decrease or increase the performance of the system. In existing literature, many statistical techniques are used for identifying efficient time quantum for RRSA. However, there is limited exposure in existing literature on generating a learning model for identifying optimized TQ. In this research work, a new research direction is given for identifying Optimized TQ by training a learning model and predicting optimum TQ value.

Thus, a new Optimized Round Robin (ORR) CPU Scheduling Algorithm is proposed for time-sharing operating systems by generating the knowledge base of feature set. The ORR is experimentally compared with RRSA and five other improved versions of RRSA. The experimental results show that ORR outperforms in terms of minimizing the Average Waiting Time (AWT), Average Turnaround Time (ATAT),

Number of Context Switch (NCS) and maximizing the throughput of the system.

KEYWORDS

CPU scheduling, multiple linear regression, operating system, round robin scheduling, time quantum.

DSMLAI '21', August 9–12, 2021, Windhoek, Namibia

ACM ISBN 978-1-4503-8763-7/20/06…$15.00
https://doi.org/10.1145/3484824.3484917

1 Introduction

The web applications and window applications generate large number of processing. The automatic processing of Big Data needs fast computational hardware and software approach. In this context, it is interesting to note that although, the problem domain of efficiency of CPU scheduling in Time-Sharing Operating Systems (TSOS) is raised in existing literature, but there is no path – breaking research work in the associated domain. The CPU scheduling algorithms are associated with a pre-defined set of rules to execute the given processes. The existing standard CPU scheduling algorithms in literature are given as First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling and Round Robin (RR)

scheduling algorithm. Any optimized CPU scheduling algorithms ensure the efficient execution of processes in minimal time domain. These algorithms are evaluated in terms of throughput of system, waiting time, turnaround time, number of context switching for a process model.

The workflow of the Round Robin CPU scheduling algorithm, the most efficient base model for static process scheduling (Silberschatz et. al., 1991), is associated with Time Slice. Initially, the processes are available in a queue called ready queue. The Round Robin Scheduling Algorithm (RRSA) (Rasmussen et. al., 2008) selects a fixed amount of Time Slice (Time Quantum - TQ) which is allocated to every process in the ready queue. The RRSA allocates the CPU to each process for a fixed TQ only. This forced allocation and de-allocation of CPU to a process for a fixed TQ shows the pre-emptive nature of this algorithm. Very small value of TQ shall increase the number of contexts switching and hence, the latency. The large values of TQ shall enable small processes to starve and may prove to be inefficient by increasing throughput and Turn-Around Time (TAT). Thus, it is evident from the existing studies that the performance of CPU scheduling in RRSA depends upon the value of TQ (Dhamdhere et. al., 2006). To handle this problem, many academic researchers have defined the value of TQ based on statistics, heuristics and dynamic values (Nayak et. al., 2012; McGuire et. al., 2014). In this context, the major contributions of this research work are:

- Generating an expert system from synthetic and artificial toy dataset of different processes and each process raise the demand of different time-period for its completion.
- Feature generation as a grid search approach for identifying the optimized value of TQ using multiple linear regression analysis by training a machine learning model.

This manuscript is classified into different Sections. Section 2 describes the historical evolution of existing improved version of RRSA by academic researchers. Section 3 defines the proposed methodology, Optimized Round Robin CPU scheduling algorithm (ORR) and its architecture. In addition to this, the workflow of the ORR is discussed in detail. Section 4 describes the experimental results and evaluation of the ORR over the synthetic dataset and compares it with the baseline model: RRSA, Adaptive Round Robin Scheduling algorithm (ARRSA), Round Robin Remaining Time algorithm (RRRT), improved Round Robin algorithm (IRR), an improvement on the improved Round Robin CPU scheduling algorithm (AAAIRR), an Enhanced Round Robin CPU scheduling algorithm (ERR). Also, the performance is evaluated in terms of increased throughput. Section 5 gives the conclusion and future score of this research work.

2 Related Work

After extensive literature survey, it is observed that the selection of the TQ value is a major challenging task for basic RRSA as it directly affects the performance of basic RRSA. The existing literature for identifying the Time Quantum (TQ) value usually focuses on the static or heuristic methodology. Since, Round- Robin scheduling algorithm is found to be the most frequently used algorithm for scheduling (Silberschatz et. al., 2019), many deterministic static TQ values are given by academic researchers. The basic RRSA was used to schedule the process which is ready to execute on the processor based on the time-sharing phenomenon. (Dhamdhare et. al., 2006). A static value of 25 was set as minimal value of TQ for multiple processes (Matarneh, 2009). This value does not ensure the effectiveness over small processes and may prove to be time consuming process.

The problem of static value of TQ was resolved by introducing an improved Round Robin (iRR) scheduling algorithm (Mishra et. al., 2010) which is based on the random TQ and handling process according to its residual time. The idea of random TQ was unique, but it was argued to be less interpretable to be used in industry. Another approach of identifying TQ was proposed as Adaptive Round Robin (ARR) scheduling algorithm (Hiranwal et. al., 2011) based on odd or even number of processes. The TQ is determined by calculating mean for even number of process and median for odd number of processes. Another modification over generating random TQ value was made by arranging process in the Shortest Job First (SJF) manner (Ramakrishna et. al., 2013).

Further, to ensure more integrity of TQ towards RRSA, the TQ was set as the execution time required by the process with minimum timespan, the reciprocal of the averaged harmonic mean of execution time of all the processes (Kishore et. al., 2014), and some modifications in iRR as an improvement on the improved Round Robin CPU scheduling algorithm (AAAIRR).

The AAAIRR made an amendment in the policy by using the hybid form of improved time allocation in every pass and the mean value of execution time of all the processes. To improve this further, a simple assumption of averaged mean of execution time of the processes and highest execution time are used for calculating TQ (Tyagi et. al., 2015). This methodology was further advanced with a proposed metric given in (RRRT) (Sharma et. al., 2015).

A comprehensive study over the improved and modified version of RRSA (ElDahshan et. al., 2017) have compared RRSA, ARR, RRRT, AAAIRR, ERR. In addition to this, some more modifications were used for comparison, for instance, Min-Max Dispersion Measure (MMRR), an Improved Round Robin algorithm with Varying Time Quantum (IRRVQ), Average Max Round Robin (AMRR). Recently, TQ is calculated by using Artificial Neural Network (Zouaoui et al., 2019) which has give a new research direction in this area of determining TQ using computational intelligence techniques. Recently, many

dynamic algorithms have also proposed for cloud computing (Alhaidari et al., 2021), and using k-means (Mostafa et. al., 2020). Thus, there is need to focus on identifying the optimized value of TQ for which the performance evaluation metrics are improved.

3 Proposed Method

The value of Time Quantum in existing literature has always been a major challenge while using the Round Robin CPU Scheduling Algorithm (RRSA). After many observations, we have proposed to formulate a new framework to identify the optimized value of the Time Quantum. This optimized value is evaluated based on many different features which are obtained as the set of features (X). To extract features, three steps are followed, namely, document organization, calculating list of number of cycles for each process, and setting parameters for feature set extraction which are discussed in this Section. To propose this framework, some assumptions are made stating that all processes that are available in ready queue are independent, have arrived on the same time in ready queue, and time is in milliseconds. The feature extraction phase is then followed by building a new multiple linear regression analysis-based model for identifying the optimized value of Time Quantum (TQ). The architecture of the framework for the proposed methodology of Optimized Round Robin (ORR) is well represented in Fig. 1.

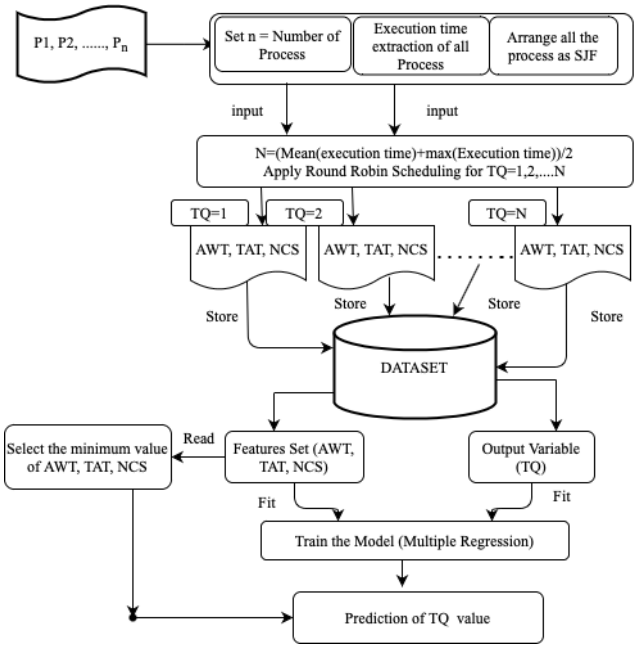

Fig 1: Architecture for the proposed model

The framework of ORR is initialized with n number of processes which will contain the information about the execution time and arrival time of each process.

3.1 Sorting the dataset

According to the increasing execution time of the processes, all the processes are organized into ascending order. This stage is the initialization stage which is known as the Shortest Job First (SJF) as shown in Algorithm 1. This arrangement shall help in determining the mode of the number of processes which is used for limiting the range to calculate the Time Quantum value.

Algorithm 1: Preprocessing dataset

1. $List\ of\ process\ P = [], Process\ Execution\ time\ \xi(P) = dict(), Arrival\ time\ \alpha(P) = dict()$
2. $Sorted\_tuple = [\,]$
3. $Temp\_list = P$
4. ***For*** $each\_Process\ in\ P$ ***do***
5. $\quad temp\_var = \xi(each\_Process)$
6. $\quad temp\_process = each\_Process$
7. For each_proc_temp in temp_list do
8. $\quad\quad$ If $temp\_var < \xi\left(each_{proc_{temp}}\right)$ ***do***
9. $\quad\quad\quad$ Continue
10. $\quad\quad$ Else:
11. $\quad\quad\quad temp\_var = \xi(each\_proc\_temp)$
12. $\quad\quad\quad temp\_process = each\_proc\_temp$
13. $\quad\quad$ End if
14. $\quad Sorted\_tuple.append(< temp\_process, \xi(temp\_process), \alpha(temp\_process) >)$
15. $\quad remove\ temp\_process\ from\ temp\_list$
16. $\quad$ End For
17. End For
18. $return\ Sorted\_tuple$

As observed from Algorithm 1, after preprocessing the dataset into a sorted list of tuples $< P, \xi(P), \alpha(P) >$ which are obtained from the synthetic and artificial dataset generated to create the set of processes, is given as an input for setting the parameters for feature extraction.

3.2 Setting parameters for Feature Set extraction

The sorted list of tuples $< P, \xi(P), \alpha(P) >$ is given as an input to calculate the number of cycles which each process will take for each TQ. The key idea is to extract features for all the processes at each TQ for a grid search approach. However, it is observed that the number of iterations for all these values shall give high complexity and thus, experiments were carried out to

generate a heuristic approach of identifying the small value of TQ to reduce the time complexity. To limit the range, the value of $limit_{range}$ is calculated as shown in Equation 1.

$$\boldsymbol{M = \frac{\sum_{i=1}^{n} \xi(P_i)}{N}} \ \boldsymbol{and}\ \boldsymbol{M' = \max\left(\bigcup_{i=1}^{n} \xi(P_i)\right)}$$

$$\boldsymbol{limit_{range} = (M + M')/2}$$

Where M is defined as the mean of the execution time of all the processes and M' is defined as the max value of the execution time among the set of all the processes. The limit of the range is defined as the average of both M and M'.

Algorithm 2: Setting Parameters for Feature set extraction

1. $Input: Sorted_{tuple}; Ouput: K = Matrix(2,2)$
2. //Calculating the number of cycles for each process
3. $limit_{range} = \frac{(mean(\xi(P)) + \max(\xi(P))}{2}$
4. **For** $each_{tuple}$ *in* $Sorted\_tuple$ **do**
5. **For** TQ *in range* $[1, limit_{range})]$ **do**
6. **If** $\frac{\xi(each_{tuple})}{TQ}$ $mod\ TQ = 0$ **then**
7. $K[each_{tuple}].append(\frac{\xi(each_{tuple})}{TQ})$
8. **Else**:
9. $K[each_{tuple}].append(\frac{\xi(each_{tuple})}{TQ} + 1)$
10. **End if**
11. End for
12. End for
13. //Setting parameters for feature set extraction
14. **For** *j in RRSA* **do**
15. **For** *i in Sorted_tuple* **do**
16. *If* $\xi(i, j-1) < TQ$: $RT(i, j-1) = \xi(i, j-1)$
17. *Else*: $RT(i, j-1) = 0$
18. $\xi(i, j) = \xi(i, j-1) - RT(i, j-1)$
19. **End For**
20. **End For**

The parameters for feature set extraction are obtained as shown in Algorithm 2. To obtain the list of the number of cycles for each process, for every TQ, a 2 X 2 matrix is obtained as $K[i][j]$ where i is each process and j is the number of cycles required for the given TQ.

$$\boldsymbol{K[i][j]} = \begin{bmatrix} \boldsymbol{k_{11}} & \cdots & \boldsymbol{k_{1L}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{k_{n1}} & \cdots & \boldsymbol{k_{nL}} \end{bmatrix}$$

where n is the number of processes for which the value of i varies and L is limit of the range of TQ for which the value of j varies. For further calculations the remaining time (RT) of the execution time of process $\xi(P)$ is given in Algorithm 2. The remaining execution time is calculated based on the remaining time and execution time of the process in the previous iteration.

3.3 Feature Extraction

There are three significant features which can be calculated from the given dataset, namely, Average Waiting Time (AWT), Turn-Around Time (TAT) and the Number of Context Switching (NCS). The features are extracted as shown in Algorithm 3.

Algorithm 3: Feature set extraction

1. $Input: Sorted_{tuple}, awt = dict, tat = dict, ncs = dict$,
2. $limit_{range} = \frac{(mean(\xi(P)) + \max(\xi(P))}{2}$
3. **For** TQ *in range* $[1, limit_{range})]$ **do**
4. $wt_{each} = dict; tat_{each} = dict, ncs_{each} = dict$
5. **For** $each_{tuple}$*in* $Sorted_{tuple}$ **do**
6. $wt_{each}[each_{tuple}] = AWT(each_{tuple}(TQ))$
7. $tat_{each}[each_{tuple}] = TAT(each_{tuple}(TQ))$
8. $ncs_{each}[each_{tuple}] = NCS(each_{tuple}(TQ))$
9. $awt[TQ] = \frac{\sum wt_{each}.values()}{len(wt_{each}.keys())}$
10. $tat[TQ] = \frac{\sum tat_{each}.values()}{len(tat_{each}.keys())}$
11. $ncs[TQ] = \frac{\sum ncs_{each}.values()}{len(ncs_{each}.keys())}$
12. **end For**
13. **end For**
14. *return* awt, tat, ncs

Initially, all the features are extracted for each process and added to three separate dictionaries. Finally, the features are averaged for each Time Quantum and obtained as

$$\boldsymbol{awt_{ij}} = \begin{bmatrix} < \boldsymbol{awt_{11}} > & \cdots & < \boldsymbol{awt_{1L}} > \\ \vdots & \ddots & \vdots \\ < \boldsymbol{awt_{n1}} > & \cdots & < \boldsymbol{awt_{nL}} > \end{bmatrix}$$

$$\boldsymbol{tat_{ij}} = \begin{bmatrix} < \boldsymbol{tat_{11}} > & \cdots & < \boldsymbol{tat_{1L}} > \\ \vdots & \ddots & \vdots \\ < \boldsymbol{tat_{n1}} > & \cdots & < \boldsymbol{tat_{nL}} > \end{bmatrix}$$

$$\boldsymbol{ncs_{ij}} = \begin{bmatrix} < \boldsymbol{ncs_{11}} > & \cdots & < \boldsymbol{ncs_{1L}} > \\ \vdots & \ddots & \vdots \\ < \boldsymbol{ncs_{n1}} > & \cdots & < \boldsymbol{ncs_{nL}} > \end{bmatrix}$$

$$\boldsymbol{X_{ij} = \{awt_{ij}, tat_{ij}, ncs_{ij}\}}$$

where X_{ij} represents the set of features. The feature set which is extracted from the dataset for every process at every TQ is then used for deterministic model to identify the optimized value of Time Quantum.

3.4 Modeling the Framework

The deterministic framework is proposed to identify the optimized value of the TQ using the feature set which is given as input. The feature set which are used for this model has the labeled dependent value of TQ. Multiple regression models describe how a single response variable Y which is taken as TQ, depends linearly on several predictor variables X which is the set of features extracted and is taken as $\{awt_{ij}, tat_{ij}, ncs_{ij}\}$ in this research work. Thus, the framework in terms of the multiple linear regression can be modeled as

$$\boldsymbol{\theta_i = a_0 + a_1 TQ_i + a_2 awt_i + a_3 tat_i + a_4 ncs_i + \epsilon}$$

The parameter ϵ is the residual terms of the model which are not considered for modeling this framework. The regression coefficients are given as the set of $\{a_0, a_1, a_2, a_3, a_4\}$ which can be trained over the multiple features set as shown in Algorithm 4.

Algorithm 4: Calculating the value of TQ

21. $Input: X = \{awt, tat, ncs\}$ // Feature Set
22. $limit_{range} = \frac{(mean(\xi(P)) + \max(\xi(P))}{2}$
23. **for** i *in range* $[1, limit_{range})]$ **do**
24. $Y = TQ_i$
25. $\theta_i = a_0 + a_1 TQ_i + a_2 awt_i + a_3 tat_i + a_4 ncs_i + \epsilon$
26. $\varepsilon_i = Y - \theta_i$
27. $re-$ *calculating* $a_0, a_1, a_2, a_3, and\ a_4, Reducing\ error\ \varepsilon_i$
28. **end for**
29. *return the_modelled_line_of_regression*

The predicted value and residuals are calculated using actual value of TQ_i and calculated value of θ_i. The residuals are then used to calculate the error sum of squares which is an important statistic referred to as a coefficient of determination. The model is then trained for the regression variables and using this trained model, the predictions are made for the best TQ which can give the least Average waiting time, the reliable turnaround time and least number of the context switches. The experiments and evaluations are made in upcoming Section and compared with existing techniques.

4 Experiments and Evaluation

The experiments are performed for identifying Time Quantum by considering an artificial dataset for this study. Since, this deterministic approach is based on the grid search; the feature set is extracted along with the labeled output of Time Quantum. The results are obtained for Optimized Round Robin CPU scheduling algorithm and compared with RRSA, Adaptive Round Robin Scheduling algorithm (ARRSA), Round Robin Remaining Time algorithm (RRRT), improved Round Robin algorithm (IRR), an improvement on the improved Round Robin CPU scheduling algorithm (AAAIRR), an Enhanced Round Robin CPU scheduling algorithm (ERR).

4.1 Case Study 1

The working instance for the proposed framework of ORR is discussed in detail. The sample of processes is taken as the set of 5 processes as {P1, P2, P3, P4, P5} which have the execution time as {24, 40, 5, 12, 34}. The processes are then arranged as per the Shortest Job First policy and hence are re-organized as {P3, P4, P1, P5, P2} with execution time array as {5, 12, 24, 34, 40}. For ORR, the defined limit of the range of TQ value is computed as 31. The resulting feature set for each value of TQ is calculated. The results show that as the value of TAT increases with TQ and so, can be defined as a beneficiary attribute. The values of AWT and NCS decreases with increase in the TQ and thus, are marked as non- beneficiary attributes. Similarly, there are six more such simulations which are considered in existing comprehensive survey (El Dahshan et. al., 2017) and are used for comparative analysis in this Section.

As there is no specific order, it is difficult to predict the Time Quantum manually. Thus, the statistical analytical methodology of multiple linear regressions is used for identifying the optimum value of TQ which is predicted as 26. The 40% of the dataset is taken as the testing dataset. The experimental setup for this research work consists of the hardware requirements of CPU @ 2.90 GHz with Intel Core i7-7500 CPU over 64-bit Operating System having 8.00 GB RAM. The software which is used for experimental results and evaluation is Python 3.

To find the optimized value of the TQ, the three attributes are taken as the deciding factor for identifying the best Time Quantum among all the given alternatives for each process. This framework contains the regression parameters {a_1,a_2,a_3,a_4} which are trained in such a way that the TQ value is optimized for minimal AWT, TAT, and NCS. The error sum of squares is calculated as

$$SSE = \sum_{i=1}^{n} (Y_i - \theta_i)^2$$

The SSE error shows that how well the model is fit over the dataset and how well are the results obtained for the given problem. Thus, the performance of the proposed framework is evaluated using SSE. However, it cannot be directly compared with the existing techniques as the existing techniques are statistical and does not use any learning-based methodology. To test and validate the robustness of the proposed approach, the predicted value of TQ is used to calculate the AWT, TAT and NCS. Similarly, the value of TQ determined in the existing techniques is used to calculate the values of AWT, TAT and NCS. The results are shown in Table 1 for seven different datasets for comparative analysis of existing techniques with the proposed method. The random value of initial TQ is taken as same for all the existing models using random value which came out to be 15 and set an important parameter for calculating Time Quantum as shown in Table 1.

Table 1: Comparative results of the existing versions of RRSA for identifying the value of TQ.

Algo_Name	AWT	TAT	NCS	TQ	Simulation 1
ORR	**32.8**	**55.8**	**7**	26	Process / CPUtime: P1 24; P2 40; P3 5; P4 12; P5 34
AAAIRR	36.2	59.2	8	12	
ARR	**32.4**	**55.4**	**7**	24	
ERR	43.4	66.4	8	12	
IRR	46.8	69.8	9	12	
RR	49.2	72.2	11	12	
RRRT	36.4	59.4	10	11	
Algo_Name	AWT	TAT	NCS	TQ	Simulation 2
ORR	28.8	47.4	7	20	Process / CPUtime: P1 7; P2 10; P3 20; P4 26; P5 30
AAAIRR	**26.8**	**45.4**	**7**	10	
ARR	28.8	47.4	7	20	
ERR	**26.8**	**45.4**	**7**	10	
IRR	30.8	49.4	9	10	
RR	32.8	51.4	10	10	
RRRT	30.2	48.8	9	9	

Algo_Name	AWT	TAT	NCS	TQ	Simulation 3
ORR	**38**	**65.8**	**6**	37	Process / CPUtime: P1 45; P2 36; P3 30; P4 18; P5 10
AAAIRR	51.8	79.6	7	15	
ARR	44	71.8	7	30	
ERR	68.8	96.6	7	15	
IRR	73.8	101.6	9	15	
RR	87.2	115	11	15	
RRRT	45.8	73.6	9	13	
Algo_Name	AWT	TAT	NCS	TQ	Simulation 4
ORR	48.8	80.8	7	34	Process / CPUtime: P1 15; P2 19; P3 23; P4 47; P5 56
AAAIRR	**46**	**78**	**7**	20	
ARR	51.2	83.2	9	23	
ERR	**46**	**78**	**7**	20	
IRR	**46**	**78**	**7**	20	
RR	58	90	10	20	
RRRT	45.2	77.2	8	16	
Algo_Name	AWT	TAT	NCS	TQ	Simulation 5
ORR	54.2	91	7	46	Process / CPUtime: P1 70; P2 53; P3 26; P4 20; P5 15
AAAIRR	68.4	105.2	7	25	
ARR	55.4	92.2	9	26	
ERR	87.8	124.6	7	25	
IRR	87.8	124.6	7	25	
RR	109.4	146.2	10	25	
RRRT	**48.6**	**85.4**	**8**	18	
Algo_Name	AWT	TAT	NCS	TQ	Simulation 6
ORR	54.2	91	7	46	Process / CPUtime: P1 70; P2 53; P3 26
AAAIRR	68.4	105.2	7	25	
ARR	55.4	92.2	9	26	

ERR	87.8	124.6	7	25	
IRR	87.8	124.6	7	25	
RR	109.4	146.2	10	25	
RRRT	**48.6**	**85.4**	**8**	18	

P4	20
P5	15

Algo_Name	AWT	TAT	NCS	TQ	Simulation 7
ORR	**53.6**	**89**	**6**	42	
AAAIRR	66.8	102.2	6	22	
ARR	61.2	96.6	7	38	
ERR	69.6	105	6	22	
IRR	69.6	105	6	22	
RR	96	131.4	10	22	
RRRT	63.8	99.2	9	17	

Process	CPUtime
P1	20
P2	55
P3	38
P4	24
P5	40

The experimental results for simulated dataset show that the proposed method ORR gives best results in 3 simulations among all 7 simulations. Although the existing models of AAAIRR and ERR gives comparative performance in 3 and 2 cases, respectively, and RRRT gives best performance in one of the seven cases, it is observed that there is no single model which gives outperforming results on any of the existing simulations.

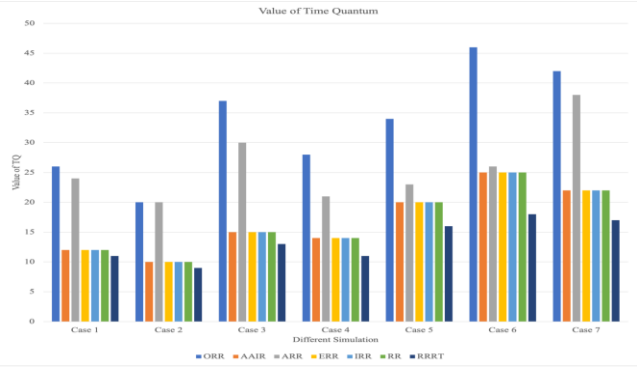

Fig 2: Time Quantum as observed for different models

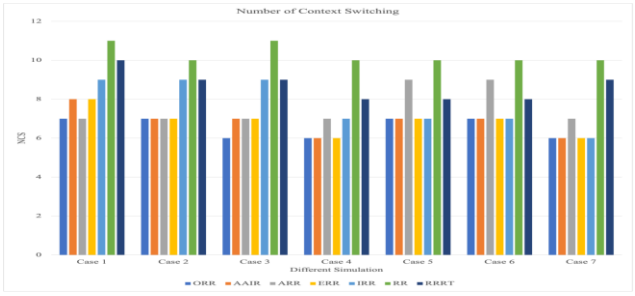

Fig 3: Context switching as observed for different models

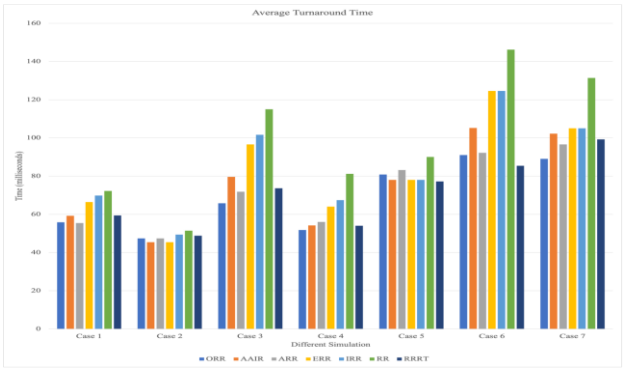

Fig 4: Turnaround Time as observed for different models

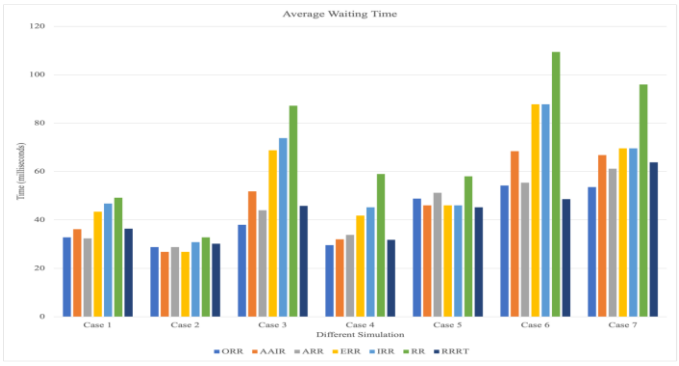

Fig 5: Average waiting Time as observed for different models

It is observed from Fig. 2 that the value of TQ by ORR is much more than which is determined by other existing models. This variation shows that the resulting value of TQ must not depend upon any static or heuristic random value for the results to be deterministic. Similarly, the number of contexts switching, as observed from Fig. 3, is found to be minimal for ORR and thus, it outperforms all the existing techniques. The resulting graphs of Fig. 4 and Fig. 5 shows that ORR gives comparatively minimal values of average TAT and AWT for any simulation dataset. We argue that none of the existing models is suitable for use over a

larger dataset in industry. To test the robustness of the proposed model and existing model, a new synthetic dataset is generated as a simulation dataset.

4.2 Case Study 3

The baselines techniques which are used to compare the results for this research paper are RR, IRR, AAAIRR, ERR ARR, RRRPT. To test the scalability of the proposed approach over multiple datasets, 100 processes are generated randomly along with random processor time. The processes are classified into training and testing dataset as 60:40 where TQ will be trained for 60% of the processes and the model will learn the values. The 40% of the processes utilize the modeled parameters to predict the value Time Quantum. The results for the seven models are observed as shown in Table 2 for this new synthetic dataset.

Table 2: Results obtained for different models for determining the value of TQ.

Algo_Name	AWT	TAT	NCS	TQ
ORR	1802.73	1851.43	126	70
AAAIRR	2342.67	2391.37	100	50
ARR	2101.31	2150.01	153	48
ERR	2342.67	2391.37	100	50
IRR	2342.54	2391.24	101	50
RR	2813.3	2862	144	50
RRRT	1934.51	1983.21	181	24

As observed from Table 2, the value of Time Quantum is different than that which is calculated by the existing techniques. The key feature of ORR is that unlike existing static, heuristic and dynamic non-deterministic algorithms, the ORR is deterministic and gives obvious results. The other importance characteristics of ORR are its scalability, reliability and gives statistically validated results.

4.3 Experimental Results

In this Section, the OOR is directly compared with the other existing models with the results given in existing literature by performing ORR over their simulations. It is observed that the results may vary for different dataset. Thus, the experiments are performed on the same dataset which is given in the existing research papers to directly compare the results.

Table 3: Direct comparison of ORR with the existing techniques for identifying the value of TQ

S. No.	Comparison	AWT	TAT	NCS	TQ
1	SATQ	62.5	112.5	6	50,25,25
	ORR	50	100	5	61
2	ERR	26.2	38.8	8	5
	ORR	18.8	31.4	7	13
3	IRR	46.2	66.8	15	10,5,5,6,6
	ORR	35.2	55.8	7	25
4	SORR	101	162.5	7	78
	ORR	99	160	7	65
5	PBRR	4	7.33	5	4
	ORR	2.33	7.33	7	5
6	MTSJ	11.3	16.66	11	3
	ORR	10.33	15.66	8	6
7	HRR	10.8	18.2	7	5
	ORR	9.8	17.2	7	10

The existing model which are considered to directly compare the proposed method are Self-Adjustment Time Quantum (SATQ) in Round Robin Algorithm Depending on Burst Time of Now Running Processes (Matarneh et. al., 2009), Efficient Round Robin (ERR) CPU Scheduling Algorithm for Operating Systems (Ramakrishna et. al., 2013), An Improved Round Robin Scheduling Algorithm with Varying Time Quantum (Mishra et. al., 2014), Smart Optimized Round Robin (SORR) CPU Scheduling Algorithm (Joshi et. al., 2015), Priority based round robin (PBRR) CPU scheduling algorithm (Zouaoui et. al., 2019), Mean Threshold Shortest Job Round Robin CPU Scheduling Algorithm (Pathak et. al., 2019), A Hybrid Round Robin (HRR) scheduling mechanism for Process Management (Ali et. al., 2020). It is observed that ORR outperforms all the existing techniques as shown in Table 3. It is well-evident from the Table 3 and other results that ORR outperforms all the existing techniques.

5 Conclusion

A new framework is introduced for identifying the optimized value of TQ which is characterized by reliability, deterministic model, and feasibility to use in industrial application because of its scalability over large number of processes. It is observed that the proposed methodology is non-heuristic and non-static method which is statistically verifiable using grid search method of identifying the optimized solution. A novel approach of multiple linear regressions is used for identifying the optimized value of TQ. The proposed method is compared with more than 10 existing models for identifying the optimized value of TQ. The results show that the ORR outperforms all the existing techniques. In future, the statistical properties of the dataset can be explored theoretically to examine and apply the ORR for industrial use.

REFERENCES

[1] Fahd Alhaidari and Taghreed Zayed Balharith. 2021. Enhanced Round-Robin Algorithm in the Cloud Computing Environment for Optimal Task Scheduling. Computers 10, 5 (2021), 63.

[2] Khaji Faizan Ali, Abhijeet Marikal, and Kakelli Anil Kumar. 2020. A Hybrid Round Robin Scheduling Mechanism for Process Management.International Journal of Computer Applications 975 (2020), 8887.

[3] S Arpita and H Gaurav. 2015. Analysis of adaptive round robin algorithm and proposed round robin remaining time algorithm. International Journal of Computer Science and Mobile Computing 4 (2015), 139–147.

[4] Dhananjay M Dhamdhere. 2006. Operating systems: a concept-based approach, 2E. Tata McGraw-Hill Education.

[5] Kamal Eldahshan, Afaf Abd Elkader, and Nermeen Ghazy. 2017. Achieving Stability in the Round Robin Algorithm. International Journal of Computer Applications 172 (08 2017), 15–20. https://doi.org/10.5120/ijca2017915161.

[6] Rahul Joshi and Sashi Bhushan Tyagi. 2015. Smart optimized round robin (SORR) CPU scheduling algorithm. International Journal of Advanced Research in Computer Science and Software Engineering 5, 7 (2015), 568–574.

[7] Samih M Mostafa and Hirofumi Amano. 2020. Dynamic round robin CPU scheduling algorithm based on K-means clustering technique. Applied Sciences 10, 15 (2020), 5134.

[8] Rami J Matarneh. 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 (2009), 1831.

[9] Christopher McGuire and Jeonghwa Lee. 2014. Comparisons of improved round robin algorithms. In Proceedings of the World Congress on Engineering and Computer Science, Vol. 1.

[10] Manish Kumar Mishra and Faizur Rashid. 2014. An improved round robin CPU scheduling algorithm with varying time quantum. International Journal of Computer Science, Engineering and Applications 4, 4 (2014), 1.

[11] Debashree Nayak, Sanjeev Kumar Malla, and Debashree Debadarshini. 2012. Improved round robin scheduling using dynamic time quantum. International Journal of Computer Applications 38, 5 (2012), 34–38.

[12] Pragati Pathak, Prashant Kumar, Kumkum Dubey, Prince Rajpoot, and Shobhit Kumar. 2019. Mean Threshold Shortest Job Round Robin CPU Scheduling Algorithm. In 2019 International Conference on Intelligent Sustainable Systems (ICISS). IEEE, 474–478.

[13] M Ramakrishna and G Pattabhi Rama Rao. 2013. Efficient Round Robin CPU Scheduling Algorithm for Operating Systems. International Journal of Innovative Technology And Research Volume 1 (2013), 103–109.

[14] Rasmus V Rasmussen and Michael A Trick. 2008. Round robin scheduling–a survey. European Journal of Operational Research 188, 3 (2008), 617–636.

[15] Dr KC SAROJ HIRANWAL. 2012. Adaptive Round Robin Scheduling using Shortest Burst Approach Based on Smart Time Slice. (2012).

[16] ARPNA SAXENA. [n.d.]. AN EFFECEINT MULTI PARAMETERIC CPU SCHEDULING ALGORITHM FOR SINGLE PROCESSOR SYSTEMS. ([n. d.]).

[17] Abraham Silberschatz, James L Peterson, and Peter B Galvin. 1991. Operating system concepts. Addison-Wesley Longman Publishing Co., Inc.

[18] Sonia Zouaoui, Lotfi Boussaid, and Abdellatif Mtibaa. 2019. Improved time quantum length estimation for round robin scheduling algorithm using neural network. Indonesian Journal of Electrical Engineering and Informatics (IJEEI) 7, 2 (2019), 190–202.