A new Improved Round Robin-Based Scheduling Algorithm-A comparative Analysis

Dr. Yosef Hasan Jbara
Computer Science Department
Buraydah Colleges
Qassim, Saudi Arabia
yosefjbara@ieee.org

***Abstract*— CPU or process Scheduling, which is an important part in any operating systems, allocates processes to the CPU in specific order to optimize some objective functions. The efficiency of any operating system relies strongly on the scheduling algorithms used. A number of scheduling algorithms exists. Among them, Round Robin (RR) is the most widely utilized algorithm. RR has proved to be effective in several types of operating systems, such as time sharing systems. This is due to the reasonable response time it gives. However, it suffers from some shortcomings such as high average turnaround time, high average waiting time as well as many context switches. Recently, several algorithms have been proposed to improve its efficiency, however, few studies were conducted to compare their efficiency when applied to dataset with different characteristics. The aim of this paper is three-fold: (1) presenting a survey of various RR based scheduling algorithms proposed and found in literature, (2) proposing a new RR based approach named, the Eighty-Five Percentile Round-Robin algorithm (EFPRR), to overcome the aforementioned issues, and (3) Conducting comparisons between eight RR based algorithms and the proposed approach using datasets with different characteristics. Extensive experiments have been done to test the proposed approach. Experimentally, the proposed approach has proven to perform better than other algorithms.**

Keywords*— *cpu scheduling, process scheduling, round robin scheduling, scheduling algorithms, average waiting time, average turnaround time, average response time, context switches.

I. Introduction

CPU or process Scheduling [1] [2] allocates processes to the CPU in specific order to optimize some objective functions. It is considered one of the most significant parts in any operating system (OS). The scheduling algorithm used affects the efficiency of the system. Several CPU scheduling algorithms have been implemented to meet various OSs requirements [1] [2], such as Round Robin (RR), Shortest Job First (SJF), Priority, First Come First Served (FCFS), and Shortest Remaining Time First (SRTF). In this work, we focus on RR, which is one of the most widely used algorithm among others.

RR is a preemptive algorithm which allocates a small amount of time called time quantum (TQ) for each ready process waiting for execution, equally and in circular fashion, treating all processes fairly without priority. It is highly preferred in time-sharing and real-time OSs [3]. This is because it allocates for each process the same amount of time to use the CPU which results in low response time [4]. However, in spite of its advantages, it suffers from high average turnaround time (ATT) [5], high average waiting time (AWT) [5], high average response time (ART) [5], and many context switches (CS) [4]. TQ plays a critical role in the behavior of the RR approach [1] [2]. Using a small value of TQ produces a lot of CSs which reduces the system performance [7] [8], while, using larger value of TQ affects negatively the ARS [6] [7].

In recent years, a number of attempts were made to improve the classical RR algorithm. All these attempts used traditional approaches and focuses around the core part of the algorithm, which is the time quantum (TQ). Some of these algorithms used static TQ and others used dynamic TQ. In addition to that, few attempts using machine learning approaches [8, 9] were made to improve the efficiency of RR algorithm via predicting the proper value of TQ.

In this paper, we survey various traditional improved RR based scheduling algorithms found in literature, and propose a new improved approach named, the Eighty-Five Percentile Round-Robin algorithm (EFPRR), to further improve the performance of RR based algorithms. Extensive experiments have been made to examine the performance of the proposed approach. Finally, this paper presents comprehensive comparisons between the proposed approach and eight RR based algorithms using ten synthetic datasets of different characteristics. To facilitate this work, a new simulator named, RR scheduling simulator, has been developed. The simulator implementation supports all RR based approaches presented in this paper.

This paper is structured as follows: Section 2 presents in brief previous work, Section 3 presents the tools used, Section 4 presents the datasets used, Section 5 presents the proposed approach, Section 6 introduces the performance measures utilized, Section 7 explains the simulation results and ameliorations analysis, and finally, Section 8 concludes the paper.

II. Literature Survey

This section provides a short review of the improved RR approaches found in the literature. This introduction is limited only to those algorithms that have been considered for comparisons in this work. The reason behind choosing these algorithms is that they are clearly presented and explained by the authors, so that they can be implemented correctly for the purpose of comparisons. Besides, there exist some other RR based algorithms in the literature which can be also implemented. But, due to time limitation, the author was not able to implement them. On the other hand, many other algorithms are existing in the literature, but, unfortunately their steps are not clear or not understandable. It is impossible to implement them. Therefore, these algorithms were not included in this section.

The authors in [10] suggested an improvement to the RR algorithm. Their approach tests the remaining time of the process under execution, if it is not greater than one TQ, the processor is re-assigned to the process for the remaining time.

978-1-5386-8125-1/19/$31.00 ©2019

The authors in [11] proposed another improvement to the RR algorithm similar to the previous one except that the currently running process will again allocate the processor if its remaining CPU time is not equal or exceeding one TQ. Jayanti Khatri [12] proposed in his paper a little improvement on the algorithm suggested by [11]. His improvement considers checking the remaining CPU time of the process whether it is not greater than one TQ. If it is, then the process is re-allocated the processor for the remaining CPU time. In their paper [13], the authors proposed an improved approach to enhance RR algorithm. Their approach decides the TQ based on the number of processes. If it is even, then the TQ equals the averages of all burst times of processes, otherwise, the TQ takes the value of the CPU time of the middle process. Arpita Sharma1 and Gaurav Kakhani [14] introduced an improved approach that calculates the TQ at the beginning of the algorithm which is equal to the summation of all processes' CPU time divided by the number of processes multiplied by two. In their paper [15], the authors proposed an improved approach that calculate the TQ dynamically at every cycle based on the difference between the maximum and minimum values of the processes' CPU times. If the calculated TQ below 25, then the TQ is set to 25, else it is set to the new calculated TQ. In [16], Manish Kumar Mishra, Faizur Rashid suggested that at each iteration the value of the TQ is set to the same value of the CPU time of the first process in the ready queue. Another introduced approach developed by [17] calculates the TQ based on the maximum value of the CPU times as well as the average CPU times. The TQ equal to the sum of average CPU times of processes and the maximum burst time divided by 2. This calculation is performed in each iteration to obtain the new TQ.

III. Tools Used

The goal of this work is to compare the performance of the proposed approach with several RR-based scheduling algorithms over different types of datasets. In order to achieve this goal, a new scheduling simulator, named RR scheduling simulator, was developed. This new simulator is advanced version of the CPU scheduling simulator developed by [18]. The simulator is provided with a number of handy functions which facilitate a comprehensive analysis of the results obtained. Examples of these functions includes: (1) creating a graphical representation of Gantt chart showing the start time and end time for each executed processes. The maximum width size of Gantt chart that can be displayed on the screen is 100 time units. The Gantt chart is created for all algorithms under comparisons, (2) displaying the results of five performance measures used to compare the algorithms. These measures are the AWT, ATT, ART, throughput, and CPU utilization. Furthermore, it displays the WT, TT and RT for each executed process, and (3) displaying the results in more interesting and readable format using charts.

IV. Datasets Used

For the purpose of fair comparisons, ten synthetic datasets were generated and used to assess the performance of the algorithms. The datasets differ in the number of processes they contain. Number of processes ranges from 50 to 200. Each process has three properties: ProcessID, arrival time and burst time. The datasets contain no noisy data. The zero arrival times and non-zero arrival times are considered. The datasets differ in their characteristics as shown in Table 1. The CPU scheduling simulator described in previous section was used to create the datasets. The randomness and distribution of these datasets are taken into consideration in order to avoid any bias in the datasets.

TABLE I. Characteristics of Datasets

	Dataset	No. of processes	Min and Max CPU Burst (sec.)	Min and Max arrival time (sec.)
1	Dataset-50Z	50	1, 14	0, 10
2	Dataset-50NZ	50	1, 12	0, 10
3	Dataset-70Z	70	1, 8	0, 15
4	Dataset-70NZ	70	1, 10	0, 15
5	Dataset-100Z	100	1, 8	0, 30
6	Dataset-100NZ	100	1, 13	0, 30
7	Dataset-150Z	150	1, 7	0, 60
8	Dataset-150NZ	150	1, 9	0, 60
9	Dataset-200Z	200	1, 8	0, 75
10	Dataset-200NZ	200	1, 10	0, 75

V. The Proposed Approach

The proposed algorithm further improves the other previously published RR based algorithms via calculating the TQ at the beginning of each step using the 85 percentile of the processes' burst time. After arranging the processes in an increasing order based on their CPU time, the TQ is computed via multiplying the mean of all processes' time with the 85 percentile constant (85%*mean). Then, it checks the remaining time of the process. If it is lesser than or equal to the calculated TQ, the processor executes the running process until completion. Or else, the process will be send to the tail of the ready queue. Figure 1 illustrated the pseudocode of the proposed algorithm. Our choice to the use of 85% came after performing a large number of experiments with the aim of obtaining a suitable constant that can be multiplied by the average burst times in order to decrease the value of the average burst times, which leads to obtaining the optimal TQ.

```
1. Assign new process to the ready queue.
2. Rearrange all processes in increasing order of their burst time
3. while ready queue is not empty
     Sum_processes_bursts = sum of all processes' burst time
     //N is number of processes in the system
     Average_burst = Sum_processes_bursts / N
     TQ = 0.85 * Average_burst
     Execute the first process in the ready queue for the calculated TQ.
     Calculate the remaining time of the current process.
     If remaining time <= TQ then
       Re-allocate the CPU to the current process for the reamining time
     Else
       Send the current processes to the end of ready queue.
       Go to step 3
     End if
     If new process arrived then
       Go to step 1
     End if
   End while
4. Calculate AWT, ATT, ART and #CS.
5. End
```

Fig. 1. Pseudu code of the proposed approach

VI. PERFORMANCE MEASURES USED

The algorithms were compared in terms of the following four performance measures [1, 2, 3].

1) Waiting Time (WT): refers to the entire time the process has to wait before its' running starts and is calculated utilizing Equation 1. W_1, W_2, …., W_n are the WTs for processes P_1, P_2, …., P_N, while N is the count of processes in the system. The performance of the algorithm increases as the value of the AWT decreases.

$$AWT = \frac{W_1 + W_2 + \cdots + W_N}{N} \quad (1)$$

2) Turnaround Time (TT): refers to the entire time from the process' submission to its accomplishment and is calculated utilizing Equation 2. TT_1, TT_2, …., TT_n are the TTs for P_1, P_2, …., P_n processes, and N represents the count of processes in the system. The efficiency of the algorithm increases as the value of the ATT decreases.

$$ATT = \frac{TT_1 + TT_2 + \cdots + TT_N}{N} \quad (2)$$

3) Response Time (RT): refers to the entire time spent between the first submission of the process to its initial output and is calculated utilizing Equation 3. RT_1, RT_2, …., RT_n are the RTs for P_1, P_2, …., P_n processes. The efficiency of the algorithm increases as the value of the ART decreases.

$$ART = \frac{RT_1 + RT_2 + \cdots + RT_N}{N} \quad (3)$$

4) Context Switches: refers to the procedure of switching the CPU from one process to another.

VII. EXPERIMENTAL RESULTS

In this section, the performance of the eight RR-based scheduling approach presented in section 2 are investigated utilizing the ten datasets described in section 4. We took into consideration three cases for arranging the incoming processes, based on their CPU times: ascending order, descending order, and random order. Extensive experiments were performed according to these orders, but, due to limited space we considered only the random order for experiments and results analysis. To give the reader an idea of how these algorithms operate, we ran all algorithms over another dataset, named dataset 1, shown in Table 2. This synthetic dataset contains only eight processes which can be easily traced. The processes have different arrival times (min = 0, max = 8) and the CPU times ranges from 1 to 9.

Employing all datasets, the final performance of the algorithms was determined by measuring AWT, ATT, ARS and #CS performance measures presented in section 6. These measures were computed for each process and the mean of each measure was calculated utilizing Equations 1, 2 and 3.

Table 3 illustrates the results using dataset 1. It is clear from the results that the proposed algorithm achieved better results in three criterions than other algorithms. It was able to produce the least values, which are 9.38, 15.25 and 7, for AWT, ATT and #CS (evenly with three other algorithms). Concerning the ART, it produced a higher value than other algorithms did. It got the sixth place among other algorithms.

TABLE II. PROCESSES DATASET 1

Process Name	Arrival Time (sec.)	CPU Time (sec.)
P1	7	9
P2	0	4
P3	4	8
P4	8	1
P5	5	2
P6	4	2
P7	6	5
P8	2	4

TABLE III. RESULTS ON DATASET 1

Algorithm	AWT (sec.)	ATT (sec.)	ART (sec.)	#CS
RR	14.25	20.12	14.25	7
IRR	14.25	20.12	14.25	7
AAIRR	11.62	17.5	5.25	11
ERR	14.25	20.12	14.25	7
ARR	13.88	19.75	7.5	11
RRRT	11.62	17.5	5.25	11
IRRVQ	16.62	22.5	2.88	25
AMRR	12.38	18.25	9	10
Yosef RR	9.38	15.25	9.38	7

Table 4 shows the simulation results of four performance measures for each scheduling algorithms over all datasets. Figure 2 shows the results graphically for all algorithms over all datasets.

The results in table 4 point out that the suggested approach achieved better results in two criterions over all datasets than other algorithms. It produced the least values for AWT and ATT. Concerning the number of CCs, it was able to produce the least number of CS evenly with three other algorithms which are RR, IRR and ERR over six datasets which are Dataset-50NZ, Dataset-70NZ, Dataset-100NZ, Dataset-150Z, Dataset-150NZ AND Dataset-200NZ. In addition to that, it was able to obtain the second place over the remaining datasets which are Dataset-50Z, Dataset-70Z, Dataset-100Z, Dataset-200Z, while, RR, IRR, and ERR got evenly the first place. With respect to the ARS, the proposed approach got the fifth place over two datasets (Dataset-50Z, Dataset-200NZ) and the sixth place over other datasets. From Figure 2, we observe the following:

1) IRRVQ produced the least value for ART over all datasets, however, it produced the highest value of CS over all datasets,

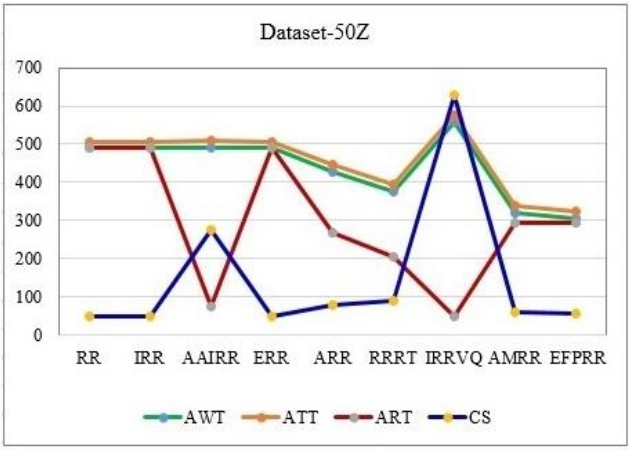
2) RR, IRR, and ERR algorithms had the same behavior regarding CS. They produced the same values of CS over all datasets.

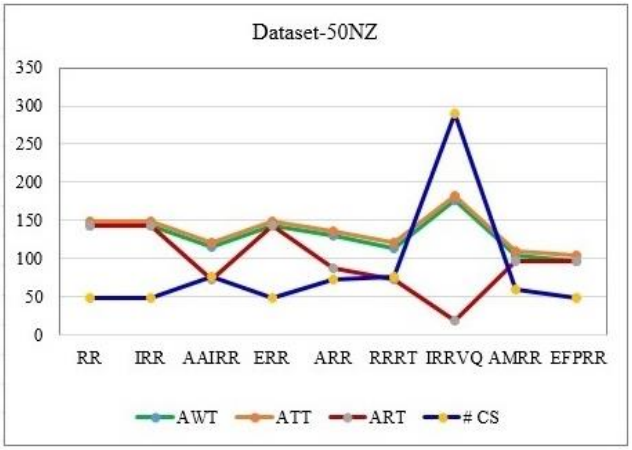
3) IRRVQ produced the highest value for ATT over all datasets.

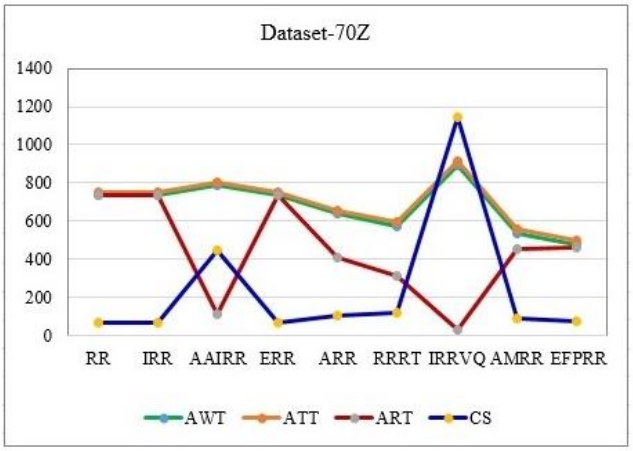
In general, the proposed algorithm proved to be the best RR based algorithm among the various algorithms applied in this paper.

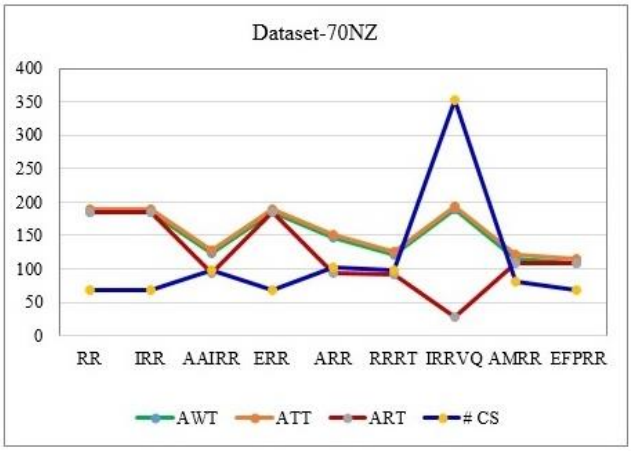
TABLE IV. " RESULTS OF EACH ALGORITHM ON ALL DATASETS

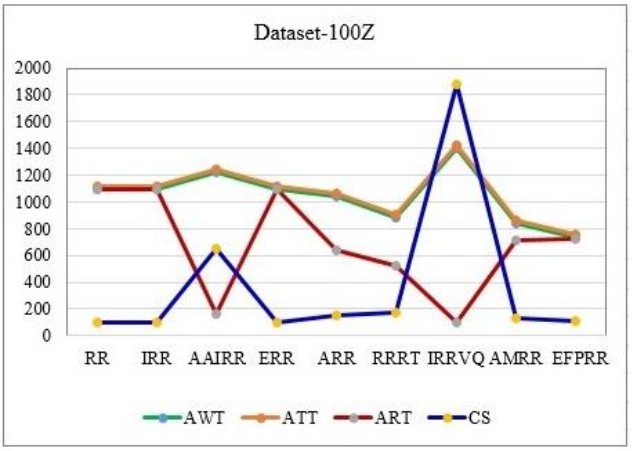
Datasets	Algorithms	AWT (sec.)	ATT (sec.)	ART (sec.)	# CS	Datasets	Algorithms	AWT (sec.)	ATT (sec.)	ART (sec.)	# CS
Dataset-50Z	RR	488.9	507.3	488.9	49	Dataset-50NZ	RR	142.5	148.3	142.5	49
	IRR	488.9	507.3	488.9	49		IRR	142.5	148.3	142.5	49
	AAIRR	489.6	508	76	277		AAIRR	115.6	121.4	73.1	76
	ERR	488.9	507.3	488.9	49		ERR	142.5	148.3	142.5	49
	ARR	425.9	444.3	267.6	79		ARR	130.7	136.5	88.4	73
	RRRT	374.1	392.4	204.5	91		RRRT	114.3	120.1	71.8	76
	IRRVQ	558.1	576.4	49	627		IRRVQ	176.7	182.5	18.4	289
	AMRR	319.5	337.9	294.6	60		AMRR	104.8	110.6	96.6	59
	Yosef RR	303.6	321.9	292.7	55		Yosef RR	97.6	103.4	97.6	49
Dataset-70Z	RR	733.1	754	733.1	69	Dataset-70NZ	RR	184.4	189.4	184.4	69
	IRR	733.1	754	733.1	69		IRR	184.4	189.4	184.4	69
	AAIRR	785.5	806.3	110	449		AAIRR	122.7	127.7	93.2	99
	ERR	733.1	754	733.1	69		ERR	184.4	189.4	184.4	69
	ARR	636.4	657.2	409.7	103		ARR	146.7	151.9	94.7	102
	RRRT	574.9	595.8	316.2	122		RRRT	120.9	125.9	91.4	99
	IRRVQ	893.7	914.6	34.5	1148		IRRVQ	189.4	194.4	28.9	352
	AMRR	536.3	557.2	456.1	94		AMRR	115.4	120.5	108.1	80
	Yosef RR	475.7	496.5	459.8	77		Yosef RR	109.1	114.2	109.1	69
Dataset-100Z	RR	1098.8	1120.2	1098.8	99	Dataset-100NZ	RR	233.3	238.5	233.3	99
	IRR	1098.8	1120.2	1098.8	99		IRR	233.3	238.5	233.3	99
	AAIRR	1225.6	1247	158.6	653		AAIRR	192.3	197.5	140	142
	ERR	1098.8	1120.2	1098.8	99		ERR	233.25	238.5	233.25	99
	ARR	1038	1059.4	636.6	154		ARR	228.1	233.3	148.6	147
	RRRT	879.7	901.1	516.7	168		RRRT	192.1	197.4	139.9	142
	IRRVQ	1397.3	1418.7	99	1875		IRRVQ	299.8	305	43.7	520
	AMRR	844.4	865.8	715.1	133		AMRR	177	182.2	170.9	111
	Yosef RR	729	750.4	723	105		Yosef RR	171.7	176.9	171.7	99
Dataset-150Z	RR	1684.6	1707.3	1684.6	149	Dataset-150NZ	RR	424.2	429.7	424.2	149
	IRR	1684.6	1707.3	1684.6	149		IRR	424.2	429.7	424.2	149
	AAIRR	2069.2	2092	232.9	1046		AAIRR	319.3	324.8	227.5	221
	ERR	1684.6	1707.3	1684.6	149		ERR	424.2	429.7	424.2	149
	ARR	1633.5	1656.3	1086.7	224		ARR	367.8	373.3	243.8	221
	RRRT	1480.2	1502.9	840.2	264		RRRT	317.5	323	228	221
	IRRVQ	2341.5	2364.3	74.5	3414		IRRVQ	496.1	501.6	69.1	823
	AMRR	1379.6	1402.3	1185.3	200		AMRR	305.8	311.3	279.1	179
	Yosef RR	1208	1230.8	1208	149		Yosef RR	282.6	288.1	282.6	149
Dataset-200Z	RR	2007	2027.1	2007	199	Dataset-200NZ	RR	574.7	580.4	574.7	199
	IRR	2007	2027.1	2007	199		IRR	574.7	580.4	574.7	199
	AAIRR	2225.8	2245.8	318	1217		AAIRR	456.4	462.1	296.8	314
	ERR	2007	2027.1	2007	199		ERR	574.7	580.4	574.7	199
	ARR	1797	1817.1	1179.7	295		ARR	508.2	514	353.9	287
	RRRT	1632.8	1652.9	951.7	346		RRRT	456	461.8	296.5	314
	IRRVQ	2582.6	2602.6	99.5	4011		IRRVQ	692.4	698.2	94.3	1149
	AMRR	1534.6	1554.6	1320.5	262		AMRR	450.6	456.4	484.5	253
	Yosef RR	1354.8	1374.9	1326	217		Yosef RR	393.4	399.1	393.4	199

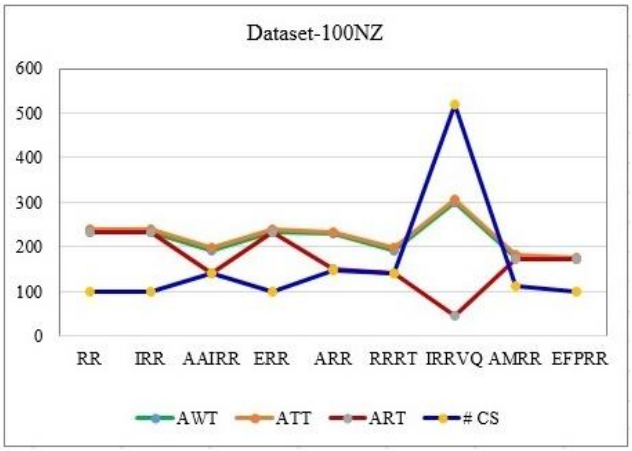


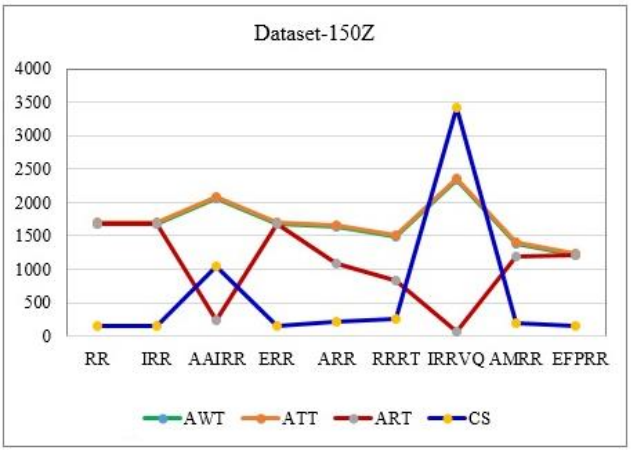


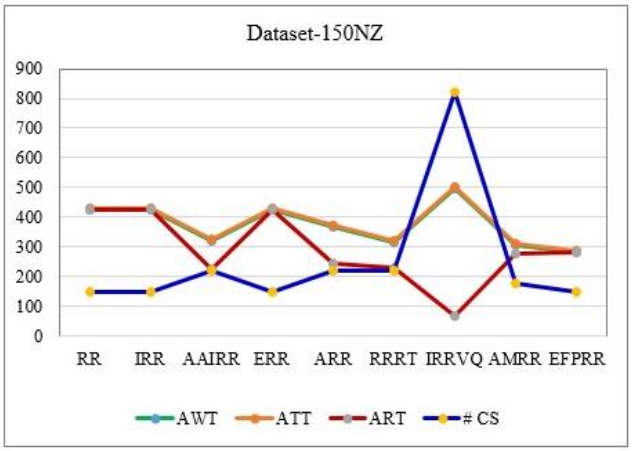












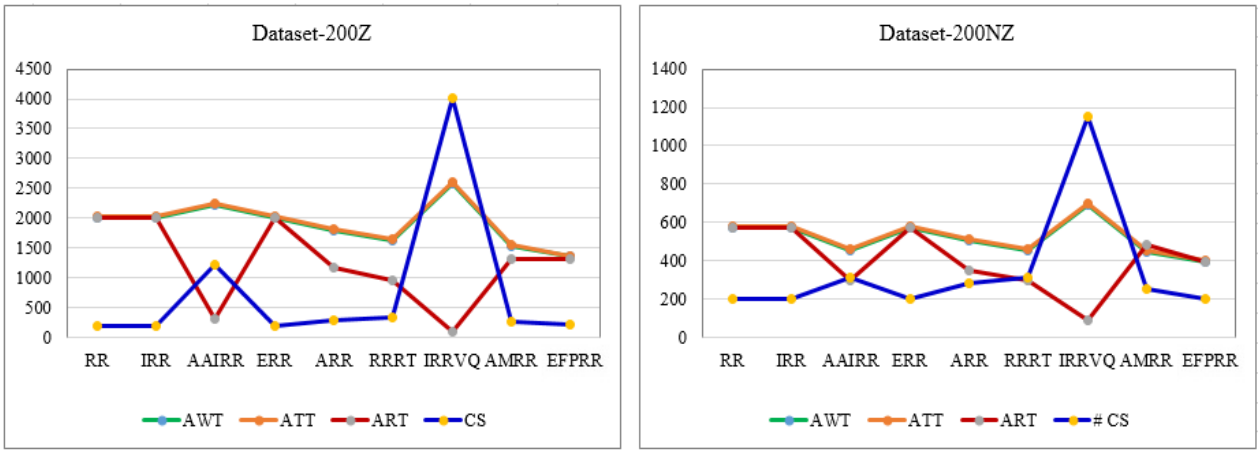


Fig. 2." Results of executing all algorithms on six datasets

VIII. Conclusion And Future Work

This paper surveys several developed approaches found in the literature to improve the performance of the classical RR scheduling algorithm. These approaches attempted to minimize the AWT, ATT, ART, and number of CSs of RR algorithm by the use of different scenarios for computing the TQ, which plays a crucial part in deciding the performance of RR algorithm. To facilitate this work, a RR scheduling simulator which supports implementation of all RR based scheduling algorithms applied in this work, was developed. Then, we proposed a new RR-based approach to further improve the efficiency of the standard RR algorithm. The proposed approach computes the time quantum based on the 85 percentile of the sum of CPU bursts of all processes'. Extensive experiments were conducted to compare the efficiency of the proposed approach against the other approaches using ten synthetic datasets each having different characteristics. The proposed approach yielded encouraging results given the difficulty of the task. It produced the best results compared to other approaches on all datasets in regard to the AWT and ATT. Concerning the ART, it produced a little bit higher values than other algorithms. Future work includes, (1) Expanding the RR scheduling simulator to support implementation of other algorithms found in literature and not considered in this work, and (2) comparing the algorithms using real-world datasets and high volume synthetic datasets.

References

[1]" S. Abraham, Peter B. Galvin and Greg Gagne, "Operating system concepts, " 9th ed., John Wiley & Sons, 2012.

[2]" S. William, "Operating systems: internal and design principles," 8th ed., Prentice Hall, Person Education, 2015.

[3]" B. Sukumar Babu, N. Neelima Priyanka and B. Sunil Kumar, "Efficient Round Robin CPU scheduling algorithm," International Journal of Engineering Research and Development, Vol 4(9), Nov. 2012, p. 36-42.

[4]" C. McGuire, J. Lee, "Comparisons of improved Round Robin algorithms", Proceedings of the World Congress on Engineering and Computer Science, Vol I, 2014.

[5]" A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating system soncepts", (9th Edn. John Wiley and Sons Inc, 2013.

[6]" Imran Qureshi, "CPU scheduling algorithms: A survey", Int. J. Advanced Networking and Applications, Vol 5(4), pp.1968-1973, 2014.

[7]" A. Muraleedharan, Neenu Antony, R. Nandakumar, "Dynamic time slice Round Robin scheduling algorithm with unknown burst time", Indian Journal of Science and Technology, Vol 9(8),pp.16, 2016.

[8]" Yosef H. Jbara, "Machine learning based multilevel feedback queue scheduling", Automatic Control and System Engineering Journal, ISSN 1687-4811, Volume 18, Issue 2, ICGST, Delaware, USA, December 2018.

[9]" S. Raheja, R. Dadhich, and S. Rajpalc: "Designing of vague logic based multilevel feedback queue scheduler", Egyptian Informatics Journal, Volume 17, Issue 1, 2016, Pages 125-137, https://doi.org/10.1016/j.eij.2015.09.003.

[10]" Manish K. Mishra, Abdul Kadir Khan, "An improved Round Robin CPU scheduling algorithm", Journal of Global Research in Computer Science, Vol 3(6), pp. 64-69, June 2012.

[11]" A. Abdulrahim, S. Aliyu, A. M Mustapha, S. Abdullahi, "An additional improvement in Round Robin (AAAIRR) CPU scheduling algorithm", International Journal of Advanced Research in Computer Science and Software Engineering, Vol 4(2), pp. 64-69, 2014.

[12]" Jayanti Khatri, "An enhanced Round Robin CPU scheduling algorithm", IOSR Journal of Computer Engineering (IOSR-JCE), Vol 18(4), pp. 20-24, Ver. II (Jul.- Aug. 2016).

[13]" S. Hiranwal, Dr. K.C. Roy "Adaptive Round Robin scheduling using shortest burst approach based on smart time Slice", International Journal of Computer Science and Communication, Vol 2 (2), pp. 319-323, (July- December) 2011.

[14]" Arpita Sharma1, Mr. Gaurav Kakhani, "Analysis of sdaptive Round Robin algorithm and proposed Round Robin remaining time algorithm", International Journal of Computer Science and Mobile Computing, Vol 4(12), pp.139-147, December 2015.

[15]" S. Kumar Panda, S. Kumar Bhoi, "An effective Round Robin algorithm using Min-Max dispersion measure", International Journal on Computer Science and Engineering, Vol 4(1), pp. 45-53, 2012.

[16]" M. Kumar Mishra, Dr. Faizur Rashid, "An improved Round Robin CPU scheduling algorithm with varying time quantum", International Journal of Computer Science, Engineering and Applications (IJCSEA), Vol 4(4), pp.1-8, August 2014.

[17]" P. Banerjee, P. Banerjee, S. Sonali Dhal, "Comparative performance analysis of average max Round Robin scheduling algorithm (AMRR) using dynamic time quantum with Round Robin scheduling algorithm using static time quantum", International Journal of Innovative Technology and Exploring Engineering (IJITEE), Vol 1(3), pp. 56-62, August 2012.

[18]" Yosef H. Jbara, "A new visual tool to improve the effectiveness of teaching and learning CPU scheduling algorithms", 2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT), 2017, DOI: 10.1109/AEECT.2017.8257759.