https://afrjmis.net

An Improved Round Robin CPU Scheduling Algorithm for Asymmetrically Distributed Burst Times

Temidayo Oluwatosin Omotehinwa[1], Igbaoreto Azeez[2] and Ezekiel Olufunminiyi Oyekanmi[3]
Department of Mathematical Sciences,
Achievers University, Owo,
Nigeria
Email: [1]oluomotehinwa@divasofttechnology.com,
[2]sdbabatunde@gmail.com,
[3]ezekny@gmail.com

ABSTRACT

The goal of Round Robin (RR) scheduling algorithm is to ensure that processes get fair share of the CPU. The efficiency in allocation is largely determined by the time quantum. Most of the existing variants of RR rely on arithmetic mean in the determination of time quantum. This is done with the assumption that burst time of processes is symmetrically distributed. However, processes may arrive into a ready queue with asymmetrically distributed burst times which are mostly associated with outliers or skewness. In this study a dynamic CPU scheduling algorithm was developed. The algorithm relies on the numeric outlier detection technique and geometric mean to determine an optimal time quantum when the burst times of processes arriving the ready queue include outliers. Some existing improved variants of RR were implemented in C++ and tested alongside the algorithm developed by this study with skewed dataset. The experimental result of the proposed algorithm was compared with the selected improved variants of RR. The result showed that the proposed algorithm performed better in terms of average waiting time and average turnaround time. In terms of number of contexts switching, only Efficient Dynamic Round Robin (EDRR) algorithm performed better than the proposed algorithm.

Keywords: *Asymmetrical Burst times, Average waiting time, Average turnaround time, Number of context switching, Improved Round Robin, CPU Scheduling, Simplified Improved Dynamic Round Robin (SIDRR).*

Reference Format:

Omotehinwa, Temidayo Oluwatosin; Azeez, Igbaoreto; and Oyekanmi, Ezekiel Olufunminiyi (2019), An Improved Round Robin CPU Scheduling Algorithm for Asymmetrically Distributed Burst Times,
Afr. J. MIS, Vol.1, Issue 4, pp. 50 – 68.

1. INTRODUCTION

The goal of Round Robin (RR) scheduling algorithm is to ensure that processes get fair share of the Central Processing Units (CPU). The efficiency in allocation is largely determined by the time quantum. The time quantum or time slice is a small unit of time allowed for a process to run before it is pre-empted by the CPU scheduler to allow other processes requesting for CPU execution time to run. It is generally between 10-100 milliseconds in length (Siberchatz, Galvin and Gagne, 2012). Most of the existing variants of RR rely on arithmetic mean (Pradhan, Behera, and Ray, 2016), (Kathuria, Singh, Tiwar, and Prashant, 2016) and other measures of central tendency. This is done with the assumption that burst time of processes are symmetrically distributed. However, processes may arrive into a ready queue with asymmetrically distributed burst times which are mostly associated with outliers or skewness. The mean of such distribution tends towards the outlier and it will no longer be representative of the dataset (Socratic, 2017). As a result, the time quantum may be too high or too low. A very low time quantum will result in high context switches which translates into the CPU being overworked. A high time quantum will create a first-come-first-serve approach and there will be increased waiting time and turnaround time (Siberchatz et. al., 2012).

In this study, a dynamic CPU scheduling algorithm was developed. The algorithm is called Simplified Dynamic Improved Round Robin (SDIRR) CPU scheduling algorithm. The algorithm relies on the Numeric Outlier Detection technique (Brownlee, 2018), (Widmann and Heine, 2018) and geometric mean to determine an optimal time quantum when the burst times of processes arriving the ready queue include outliers. The specific objectives of this study were to: i. develop an algorithm that determines the existence of an outlier in any given dataset; ii. determine an appropriate computation for Time Quantum (TQ); iii. develop an algorithm that implements i & ii and gives better results based on average waiting time, average turnaround time and number of context switches in comparison to some selected RR CPU scheduling algorithms; iv. implement the algorithm in (iii) using C++. The selected improved variants of RR; New Improved Round Robin (NIRR), Improved Round Robin with Varying time Quantum (IRRVQ), Dynamic Average Burst Round Robin (DABRR), Revamped Mean Round Robin (RMRR) and Efficient Dynamic Round Robin (EDRR) developed by (Abdulrazaq, Saleh, and Junaidu, 2014), (Manish and Rashid, 2014), (Dash, Sahu, and Samantra, 2015), (Kathuria, Singh, Tiwar, and Prashant, 2016) and (Farooq, Shakoor, and Siddique, 2017) respectively, were implemented in C++ and tested with dataset containing outliers alongside SDIRR algorithm proposed by this study.

2. RELATED WORK

(Rajput and Gupta, 2012) in their study developed an algorithm 'PBRR' that prioritised incoming process on a randomly selected time slice at first cycle. Subsequent cycle is considered on shortest job first assigned highest priorities over those processes with high burst time. CPU is assigned till execution according to the prescribed order. Their algorithm cancels out possibility of convoy effect on processes. It's possible flaw could be the arbitrary selection of TQ which can lead to high number of context switches and increase in Average Waiting Time (AWT) and Average Turnaround Time (ATT).

(Patel and Milli, 2013) coined the 'SJRR' CPU scheduling algorithm; pre-emptive in nature and based on Round Robin scheduling mechanism. It uses static selection method of assigning TQ. The smallest job is assigned as the TQ. The study dataset is too small to reflect efficiency. Also, the processes burst time is evenly distributed. High burst time processes are starved.

(Agha and Jassbi, 2013) asserted in their work that heterogeneous datasets need a more modern approach in measure of central tendency. The study introduces a Pythagorean mean family member: the harmonic mean that reciprocates the individual burst time and give a weighted mean by using it to divide the number of processes under study. According to their study, the algorithm performs best when considering heterogeneous processes with new lesser burst time than the previous. While the work defines the heterogeneous possibilities of dataset, it fails to consider its degree of variation and possible means of improving the TQ to suit the processes. The processes in the study are also too small to get a clearer degree of improvements and efficiency.

(Manish and Rashid, 2014) presented an improved Round Robin CPU scheduling algorithm. The algorithm named IRRVQ was focused on enhancing CPU performance using the features of Shortest Job First and Round Robin scheduling with varying time quantum. The proposed algorithm is experimentally proven better than conventional RR. Sorting the burst times of processes in ascending order, the algorithm combines the Shortest Job First with conventional Round Robin. At every CPU service cycle, the shortest burst time is implemented as its time quantum. Although, simulation showed remarkable improvements in reduction of average waiting time and average turnaround time compared to traditional RR, the

algorithm is susceptible to overhead due to high number of context switches. The presence of outlier (standalone high burst time) in the process will lead to a greater Number of Context Switches (NCS). There is hint of starvation to high burst times as the algorithm favours shortest job first.

This study (Abdulrazaq, Saleh, and Junaidu, 2014) proposed the 'NIRR' (New Improved Round Robin) a huge benchmark algorithm that combined the FCFS and Round Robin. With weighted average of burst times of post arrival processes after the first in the CPU, the algorithm subject remainder burst time to a conditional assessment to know if it is to pull out and return to the queue or re-access the CPU. It has two queues; one to collate arrival processes while the other holds the ready to run processes. At each cycle, the remaining processes are reconsidered along with newest arrivals, if any; the next average is computed and assigned as TQ dynamically until all jobs are completed. The implementation result obtained from the algorithm (NIRR) is preferred for systems that adopt the RR Scheduling because it produces minimal AWT, ATAT and NCS compared to other algorithms. The authors concluded that more tests should be done based on the burst time of processes that follow different patterns of statistical distributions. However, it is worthy of note that a higher burst time from such different statistical pattern will cause a pronounced convoy effect, possible starvation to jobs of smaller burst times as well as increase the AWT, ATAT and NCS. The algorithm proposed in this study is aimed at bridging the gaps highlighted above.

(Dash, Sahu, and Samantra, 2015) used the arithmetic mean to compute its TQ. Arrived processes are sorted in ascending order, at every cycle, if there is any process in the CPU service cycle with remaining burst time less than the TQ, such process is given access to CPU to complete its execution. This dynamic algorithm tends to rely solely on normal distribution with ideal unbiased data set. The study implemented a ready queue with 5 processes p1, p2, p3, p4, p5. These processes all arrived at same time (zero arrival). The burst time of p1, p2, p3, p4, p5 are 15, 32, 102, 48, 29 milliseconds respectively. First the processes are sorted in ascending order of their burst time which provides the sequence p1, p5, p2, p4, p3. The time quantum is set equal to the mean of burst time of all 5 processes i.e. 45. P1, P5, P2 get completed after executing all processes for a time quantum of 45 milliseconds. So they are removed from the ready queue. After first cycle, the remaining burst time for p3 and p4 are 3 and 57 respectively. The new time quantum is set equal to the mean of the burst time of the processes in ready queue in the next cycle i.e. 30, and CPU is assigned to the processes for the new time quantum. After the second round the process p4 has finished execution and only process p5 remains in the ready queue with burst time 27 millisecond. As only one process is there in the ready queue so its burst time is directly chosen as time quantum and CPU is allocated to p5. Accordingly, the turnaround time for p1, p2, p3, p4, p5 are 15, 76, 226, 169, and 44 milliseconds respectively. The average turnaround time is 106. The waiting time for p1, p2, p3, p4, p5 are 0, 44, 124, 121, and 15 milliseconds respectively. The average waiting time is 60.8.

(Pradhan, Behera, and Ray, 2016) developed 'Modified Round Robin (MRR)' with a focus on the priority of first arrival process. Subsequent processes are made to wait while the first process complete execution. Two registers are maintained to keep update of the sum of all burst times and equivalent average (mean) which is assigned as the TQ. Any process that does not complete its execution within the slotted TQ is moved to the end of the queue while the registers are updated for the next cycle. In comparison to the standard RR, it greatly reduces the AWT, ATAT and NCS. However, first process of very high burst time makes the algorithm susceptible to convoy effect. Also, using the arithmetic mean to compute the TQ does not adequately represent the dataset distribution as it tends towards the outlier.

(Kathuria, Singh, Tiwar, and Prashant, 2016) coined the "RMRR" algorithm. The study emphasizes the inadequate knowledge of the burst times of processes while arriving. A unique time slice of two is assigned for all processes arriving in the pre-ready queue. Subsequently, all the remaining processes are moved to the ready queue while the average burst time is allotted as the new time slice/ TQ at every cycle. Any remainder burst time less than the TQ at each cycle, is allowed access to CPU to complete its execution. The study gives adequate improvement over conventional RR. The flaw of it is that data of heterogeneous statistical distribution will affect its performances negatively. Also, it recorded high number of context switches.

(Musa, Lasisi, and Gokir, 2017) stated in their study that not all improvements are better than the previous ones. Their work understudied previous developed algorithms like Optimized RR (DABRR), Dynamic time Sliced Round Robin (DTSRR), Revamped Mean Round Robin (RMRR), Improved Round Robin with Varying time Quantum (IRRVQ), SARR, DQRRR, RP-5 and MRR with respect to standard RR. Also, this study suspects that most scholars do test their algorithm with few data set and small burst times and then conclude. However, it was observed that the dataset implemented in the study is also homogenous and un-skewed. Their results exposed the efficiency of DABRR over other algorithms like RMRR

developed later. This is why DABRR was selected for comparative analysis in this study.

EDRR is the outcome of the study by Farooq, Shakoor, and Siddique, (2017) when they proposed to implement a percentage on maximum burst time and assign it as TQ. Their study looks into minimal burst times below the TQ for execution in the CPU. Any process with burst time higher than the TQ is made to wait till the minimal one finish first. Subsequently, TQ is reassigned highest burst time and all the process run till completion. Obviously, the algorithm gives huge concession to shortest jobs; its quantum is also a reflection of Pareto (80/20% rule) statistical distribution in the data (though not adequately spelt out). There is hint of starvation to higher burst time.

Survey studies reveal the following shortfalls

1. Datasets are inadequate as most considers five number of process to conclude their study
2. Most datasets considered are homogeneous and uniformly distributed datasets.
3. Most TQ selection relies on arithmetic mean which does not adequately represent the skewness of a dataset.
4. Most study revolves around starvation and convoy effects.

The algorithm proposed by this study determines an appropriate time quantum based on the existence or non-existence of outliers. With the itemized gaps bridged the proposed algorithm should perform better.

3. METHODOLOGY

This section presents the mathematical formulations for each of the objectives stated in section 1.

3.1 Definitions and formulations

Let TT_j denote turnaround time of the j[th] process.

Let CT_j denote completion time of the j[th] process.

Let AT_j denote arrival time of the j[th] process.

Let WT_j denote the waiting time of the j[th] process.

Let NCS represent number of context switching.

Let $Q1, Q3\ and\ IQR$ be the first, third quartile and Interquartile Range respectively.

Let ω represent the number of times pre-emption takes place in the CPU.

k denotes the Interquartile range multiplier. $k = 1.5$

RQ denote the ready queue

AQ denote the set of processes

γ represents the number of process in the set under consideration.

$\beta t[i]$ denotes the burst time of a given i[th] process in the queue.

Let βt represent the set $\{x_1, x_2, \dots x_n\}$ which is a set of burst times of processes.

Suppose $x_1, x_2, \dots x_n$ are independent identically distributed random variables and represents burst times of processes whose probability distribution is supported on the interval $[x_m, \infty]$ for some $[x_m > 0]$;

Suppose for all n, the two random variables $\min\{x_1, x_2, \dots x_n\}\ and \frac{\{x_1, x_2, \dots x_n\}}{\min\{x_1, x_2, \dots x_n\}}$ are independent.

Then, the common distribution is a Pareto distribution; and has a median denoted by M;

Let $A\ and\ B$ represent the lower and upper inner fences respectively.

Let τ denote the time quantum.

ϑ represents a Boolean value therefore, if 1_ϑ there exist outlier, otherwise, false.

ζ represents geometric mean, while C represents outlier value x_i found in βt.

Given that:

$$\beta t = \{x_1, x_2, \dots x_n\}$$

Then,

$$\gamma = |\beta t|$$

3.1.1 Numeric Outlier Detection

Numeric outlier is a simple nonparametric outlier detection method. Using the computation of Interquartile Range(IQR), the first and third quartiles $Q1\ and\ Q3$ are computed (equation 3.7). An outlier is then a data point that lies outside the interquartile range. This is represented in equations (3.3) to (3.19) in the Appendix.

3.1.2 Flowchart

NIRR CPU scheduling algorithm:

Step 1: Start

Step 2: Create a queue, ARRIVE, where processes will be placed as they arrive into the system
before they are moved to the ready queue

Step 3: Create a ready queue, REQUEST

Step 4: Do

Step 5: If (process Index =1) {
Time quantum=burst time [1]
Move the first process (pr[1]) to REQUEST queue}

else {
Move all processes in ARRIVE queue to REQUEST queue in ascending burst time order
Time quantum = $\frac{\sum_{i=1}^{n} burst\_time[i]}{n}$
}
Step 6: Do
Step7: Allocate the CPU to the first process in REQUEST queue for a period of 1-time quantum.
Step 8: If the remaining CPU burst time of the currently running process is less than or equal to
half time quantum then, allocate the CPU again to the currently running process for remaining CPU burst time. After completion of execution, remove the process from the ready queue and go to step 7.
Step 9: If the remaining CPU burst time of the currently running process is longer than half time
quantum, remove the process from the REQUEST queue and put it in the ARRIVE queue
and go to step 7.
Step 10: If a new process arrives the system, it is placed in the ARRIVE queue.
Step 11: WHILE queue REQUEST is not empty.
Step 12: WHILE queue ARRIVE is not empty.
Step13: Calculate AWT, ATAT, ART and NCS.
Step 14: END

DABRR CPU scheduling algorithm:
TQ: Time Quantum
RQ: Ready Queue
n: number of process
Pi: Process at ith index
i, j: used as index of ready queue
TBT: Total Burst Time
[1] Arrange the processes in ascending order.
[2] n = number of processes in RQ
[3] i=0, TBT=0
[4] Repeat step 5 and 6 till i < n
[5] TBT += burst time of process Pi
[6] i++
[7] TQ = TBT/n
[8] j = 0
[9] Repeat from step 12 to 19 till j<n
[10] if (burst time of Pi) <= TQ
[11] Execute the process
[12] Take the process out of RQ
[13] n--
[14] Else
[15] Execute the process for a time interval up to 1 TQ
[16] Burst time of Pi = Burst time of Pi – TQ
[17] Add the process to ready queue for next round of execution
[18] j++
[19] If new process arrives
[20] goto step 1
[21] If RQ is not empty
[22] goto step 2

IRRVQ CPU scheduling algorithm:
Step 1: Make a ready queue RQUEUE of the Processes submitted for execution.
Step 2: DO steps 3 to 9 WHILE queue RQUEUE becomes empty.
Step 3: Arrange the processes in the ready queue REQUEST in the ascending order of their remaining burst time.
Step 4: Set the time quantum value equal to the burst time of first process in the ready queue RQUEUE.
Step 5: Pick the first process from the ready queue RQUEUE and allocate CPU to this process for a time interval of up to 1-time quantum.
Step 6: Remove the currently running process from the ready queue RQUEUE, since it has finished execution and the remaining burst time is zero.
Step 7: REPEAT steps 8 and 9 UNTIL all processes in the ready queue gets the CPU time interval up to 1-time quantum.
Step 8: Pick the next process from the ready queue RQUEUE, and allocate CPU for a time interval of up to 1-time quantum.
Step 9: IF the currently running process has finished execution and the remaining CPU burst time of the currently running process is zero, remove it from the ready queue ELSE remove the currently running process from the ready queue RQUEUE and put it at the tail of the ready queue.

RMRR CPU scheduling algorithm:
Step 1: Start
Step 2: Create two queues PREREADYQUEUE (PRQ) and READYQUEUE (RQ)
Step 3: new processes will put into PREREADYQUEUE
Step 4: while (PRQ is not empty)
Perform Round Robin with time quantum equals to two-unit time.
Move all processed process to RQ from PRQ.
Step 5: In READYQUEUE, apply following steps.
Step 6: TQ = mean of burst time of processes present in RQ.
Step 7: Allocate CPU to first process present in READYQUEUE.

Step 8: If the remaining CPU burst time of the currently running process is less than time quantum, then, allocate the CPU again to the currently running process for remaining CPU burst time. After completion of execution, remove the process from the ready queue and allocate CPU to next process.
Step 9: If the remaining CPU burst time of the currently running process is longer than time
quantum then allocate CPU to next process in READYQUEUE.
Step 10: If a new process arrives in the system during this time, put it into PREREADYQUEUE
and it will wait till current process gets executed completely in READYQUEUE. Now go to step 4.
Step 11: WHILE READYQUEUE is not empty.
Step 12: Calculate AWT, ATAT, ART and NCS.
Step 13: END

EDRR CPU scheduling algorithm:

```
[1] BTmax = Maximum Burst Time;
[2] BTi = Burst time of ith process
[3] QT = Quantum Time;
[4] N = Number of processes in ready queue;
[5] remainingProcesses = Remaining processes;
[6] i = 1;
[7] QT = 0.8 * BTmax
[8] while i < N do
[9]    if i < N then
[10]        if Bi <= QT then
[11]           assign CPU to the processes;
[12]           N - -;
[13]        else if Bi > QT then
[14]              don't assign CPU and put the processes at the end of the ready queue;
[15]           remainingProcesses++;
[16]   else if i == N && remaining > 0 then
[17]         QT = BTmax;
[18]              i == 0;
[19]    i ++;
[20] end
```

Figure 2. presents an illustrative Gantt chart for the processes with burst time in descending order and non-zero arrival time (See experimental settings in Table 6). The charts show the performances NIRR, DABRR, IRRVQ, RMRR, EDRR and the proposed algorithm SIDRR. The comprehensive result of this illustration is presented in Table 9.

4. EXPERIMENTAL ANALYSIS

In order to validate the correct implementation of the selected algorithms, the processes used in each paper were fed in as input into the programs written in C++ programming language for each of the selected algorithms. The program produced the same results as presented in each paper. The processes from each of the papers were also fed into the SIDRR algorithm for comparative study. The results are presented in Figures 3 – 7 (Appendix I).

The experimental analysis is based on arrival time of processes. This is divided into two: Zero arrival time and Non-zero arrival time. For each of these categories, the order of burst time of processes; ascending order, descending order and random order (Table 1) were considered. Twenty processes with arrival time and burst time were considered. The processes burst times were generated randomly.

This study assumed single processor environment, the arrival time and burst time are known before execution, and the time required to sort processes is not consequential.

4.1 Experimental Settings

The experimental settings are as indicated in Tables 2 to 7.

4.2 Results and Discussion

Tables 8, 9, and 10 present the comparative analysis results for the experimental settings in Tables 2, 3, and 4 respectively. The proposed algorithm, SIDRR has a minimum AWT and ATT in the descending and random order scenario and equal performance with EDRR in the ascending order scenario. However, EDRR has a smaller NCS.

In the non-zero arrival category (Tables 5 - 7), the result of the experimental simulation presented in Tables 11 – 13, it can be observed that the proposed algorithm also performed better with minimum AWT and ATT. EDRR has the minimum NCS in comparison to SIDRR in the three cases (ascending, descending and random order).

4.2.1 Comparative Analysis

The summary of the performance of the six algorithms based on number of context switches, average waiting time, and average turnaround time is presented in Table 14 - Table 16.

Considering the IRRVQ as the baseline improvement over the conventional RR, the comparative analysis results affirms the proposed algorithm SIDRR, is an improvement over most of the selected existing works.

The results in Table 17 and 18 showed the percentage of average waiting time and average turnaround time saved by each algorithm than Improved Round Robin Varying time Quantum algorithm. The proposed algorithm "SIDRR" saved 47.05% waiting time and 38.81% as compared with IRRVQ.

5. CONCLUSIONS

This paper presents a modification of Round Robin scheduling algorithm. Comparative analysis of various algorithms like IRRVQ, DABRR, RMRR, EDRR, NIRR and the proposed algorithm SIDRR has been done. It can be observed from the results obtained from the six categories that most of the selected algorithms have a high initial time quantum as a result of the presence of outlier. Only SIDRR determined a relatively optimal initial time quantum even with the presence of outlier. In terms of number of context switching, only Efficient Dynamic Round Robin (EDRR) algorithm performed better than the proposed algorithm.

REFERENCES

[1] Pradhan, P Behera, P. K., and Ray, B. N. B. (2016). 'Modified Round Robin Algorithm for Resource Allocation in Cloud Computing', *Procedia Computer Science,* Vol. 85, pp. 878 – 890.

[2] Kathuria, S., Singh, P. P., Tiwar, P. I., and Prashant. (2016). 'A Revamped Mean Round Robin (RMRR) CPU Scheduling Algorithm', *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 4, Issue 4, pp. 6684-6691.

[3] Socratic, (2017). How does an outlier affect the mean of a dataset? Retrieved May 15, 2019 from https://socratic.org/questions/ how-does-an-outlier-affect-the-mean-of-a-data-set

[4] Silberschatz, A., Galvin, P. B., and Gagne, G. (2012). *Operating System Concepts* (9th ed.). New Jersey, United States of America: John Wiley & Sons, Inc.

[5] Brownlee, J. (2018). How to use statistics to identify outliers in data. Retrieved May 15, 2019 from https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data

[6] Widmann, M. and Heine, M. (2018). Four techniques for outlier detection. Retrieved May 15, 2019 from https://www.knime.com/blog/ four-techniques-for-outlier-detection

[7] Abdulrazaq, A., Saleh, E. A., and Junaidu, S. B. (2014). 'A New Improved Round Robin (NIRR) CPU Scheduling Algorithm', *International Journal of Computer Applications,* Vol. 90, Issue 4, pp. 27-33.

[8] Manish, K. M., and 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, Issue 4, pp. 1-8.

[9] Dash, A. R., Sahu, S. k., and Samantra, S. K. (2015). 'An Optimized Round Robin CPU Scheduling Algorithm with Dynamic Time Quantum', *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT),* Vol. 5, Issue 1, pp. 7-26.

[10] Farooq, M. U., Shakoor, A., and Siddique, A. (2017). An Efficient Dynamic Round Robin Algorithm for CPU Scheduling. *IEEE International Conference on Communication, Computing and Digital Systems* pp. 244-248.

[11] Rajput, I. S. and Gupta, D. (2012). 'A Priority based Round Robin CPU Scheduling Algorithm for Real Time Systems', *International Journal of Innovations in Engineering and Technology (IJIET),* Vol. 1, Issue 3, pp. 1-11.

[12] Patel, R. and Milli, P. (2013). 'SJRR CPU Scheduling Algorithm', *International Journal of Engineering and Computer Science,* Vol. 1, Issue 12, pp. 3396-3399.

[13] Agha, A. E. A. and Jassbi, S. J. (2013). 'A New Method to Improve Round Robin Scheduling Algorithm with Quantum Time Based on Harmonic-Arithmetic Mean (HARM)', *Modern*

Education and Computer Science, Vol. 5, Issue 7, pp. 56-62.

[14] Musa, K. I., Lasisi, K. E., and Gokir, J. A. (2017). 'Comparative Performance Analysis of Some Improved Round Robin CPU Scheduling', *International Journal of Scientific & Engineering Research*, Vol. 8, Issue 8, pp. 1494-1502.

Biodata

Omotehinwa, Temidayo Oluwatosin obtained his M.Sc. in Computer Science in 2011 and his Ph.D. from the Department of Computer Science, Faculty of Communication and Information Sciences, University of Ilorin, Ilorin, Nigeria in 2017. He is currently a Senior Lecturer with the Department of Mathematical Sciences, Achievers University Owo, Ondo State, Nigeria. He is a Rank Sheet Certified Programmer. His research interest includes Machine Learning, Web Programming, Embedded Systems, Internet of Things and Cloud Computing. He has authored and/or co-authored more than 15 articles in refereed Journals and conference proceedings. Dr. Omotehinwa is a member of Nigeria Computer Society (NCS) and an associate member of the Institute of Strategic Management (ISMN).

Azeez, Saeed Igbaoreto is an engineer cum Scientist. Azeez holds an M. Sc in Computer Science. from Achievers university, Nigeria. He received HND certificate in Electrical and Electronics Engineering (Power option) from Federal Polytechnic Ado Ekiti, Nigeria and the B. Sc degree in Computer science from Achievers University, Nigeria. His research interests focus on Machine Learning, Internet of Things, Smart Grid, Economic Load Dispatch automation and Optimization for generation systems in the energy Sector. He is a member of Institute of Safety Professionals of Nigeria (ISPON).

Oyekanmi, Ezekiel Olufunminiyi (Ph. D.) is a Lecturer I in the Department of Mathematical Sciences, College of Natural and Applied Sciences, Achievers University, Owo, Ondo State, Nigeria. His research interests include: Image Processing, Data Analysis, and High-Performance Computing.

APPENDIX

3.1.1 Numeric Outlier Detection

$$\forall x_i \in \beta t > Q3 + k(IQR) \vee \forall x_i \in \beta t < Q1 - k(IQR) \quad (3.3)$$

$$IQR = Q3 - Q1 \quad (3.4)$$

$$A = Q1 - k(IQR) \quad (3.5)$$

$$B = Q3 + k(IQR) \quad (3.6)$$

$$f(\gamma) = \begin{cases} M = \frac{\gamma+1}{2} term & \gamma \text{ is an odd} \\ M = \frac{\left[\left(\frac{\gamma}{2}\right)^{th} term + \left\{\left(\frac{\gamma}{2}\right)+1\right\}^{th} term\right]}{2} & \gamma \text{ is an even} \end{cases} \quad (3.7)$$

when γ is even,

$Q1 = median\ of\ the\ \gamma\ smallest\ entries$

$Q3 = median\ of\ the\ \gamma\ largest\ entries$

When γ is odd,

$Q1 = median\ of\ the\ \gamma\ smallest\ entries$

$Q3 = median\ of\ the\ \gamma\ largest\ enries$

The Interquartile Range (IQR) is computed using equation (3.4) while the inner fences A (lower boundary) and B (upper boundary) are generated from equation (3.5) and (3.6)

Check for outlier,

$$\forall x_i \in \beta t > A \vee \forall x_i \in \beta t < B \ \exists \ C \quad (3.8)$$

Hence, 1_ϑ otherwise $\vartheta = 0$

Given geometric mean:

$$\zeta = \left(\prod_{i=1}^{\gamma} \beta t[i] \right)^{1/\gamma} \quad (3.9)$$

To compute appropriate ***time quantum*** τ when 1_{ϑ},
$if\ \gamma = 1\ then,$

$$\tau = \beta t[i] \quad (3.10)$$

If $C > Q3\ and\ \gamma > 1\ then,$

$$\tau = Q3 \quad (3.11)$$

Otherwise,

$$\tau = \zeta \quad (3.12)$$

When ϑ' indicating non-existence of outliers
$if\ \gamma = 1\ then,$

$$\tau = \beta t[i] \quad (3.13)$$

Otherwise;

$$\tau = \zeta + \left(\frac{Q1}{2} \right) \quad (3.14)$$

In this study the selected comparison parameters are average waiting time (Eqn. 3.17), average turnaround time (Eqn. 3.18) and number of context switching (Eqn. 3.19).

$$TT_j = CT_j - AT_j \quad (3.15)$$

$$WT_j = TT_j - \beta t[i] \quad (3.16)$$

Hence, the average waiting time is:

$$AWT = \frac{1}{\gamma} \sum_{j=1}^{\gamma} WT_j \quad (3.17)$$

the average turnaround time is:

$$ATT = \frac{1}{\gamma} \sum_{j=1}^{\gamma} TT_j \quad (3.18)$$

The number of context switching is determined thus;

$$NCS = \sum_{i=1}^{\gamma} (\omega_i - 1) \quad (3.19)$$

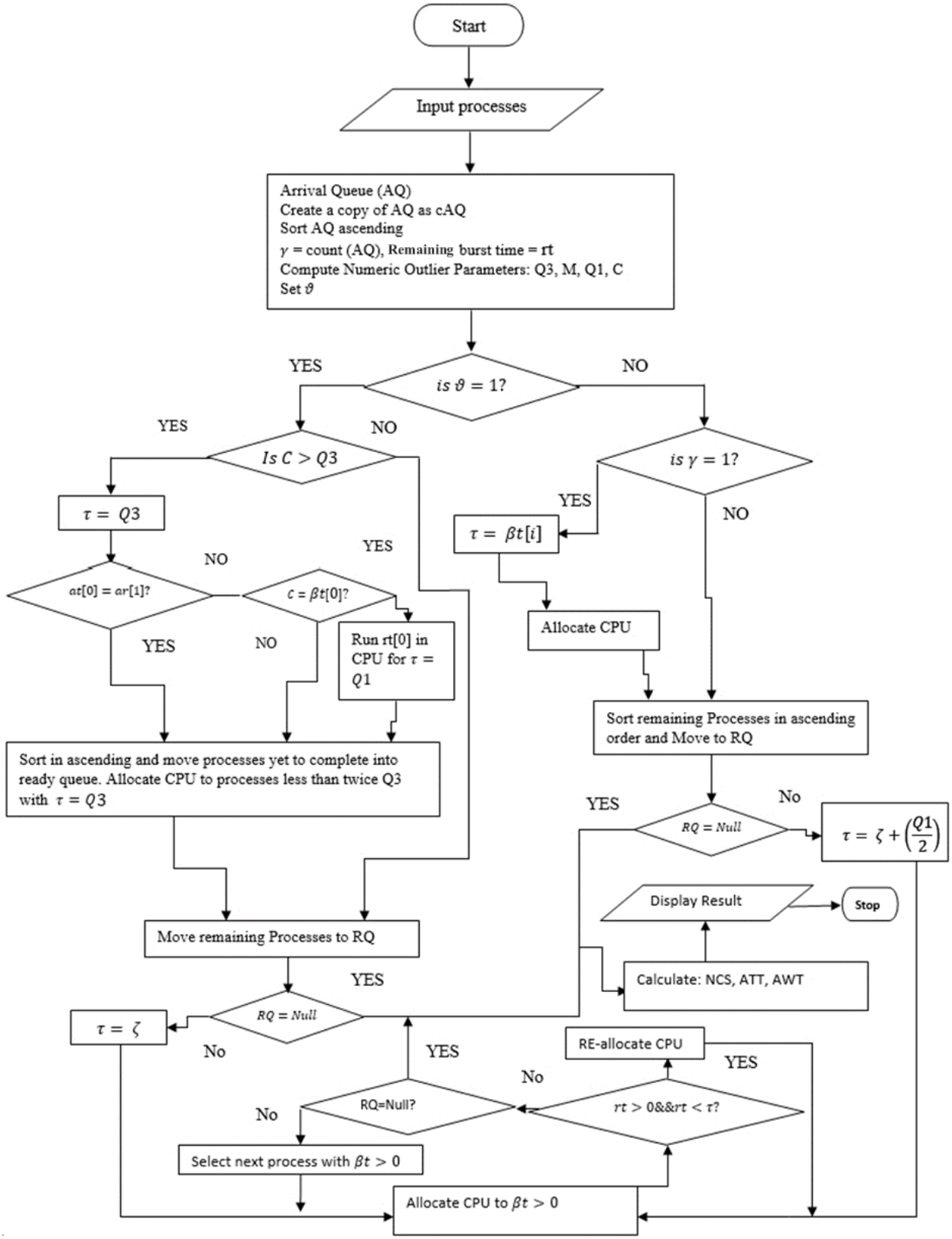
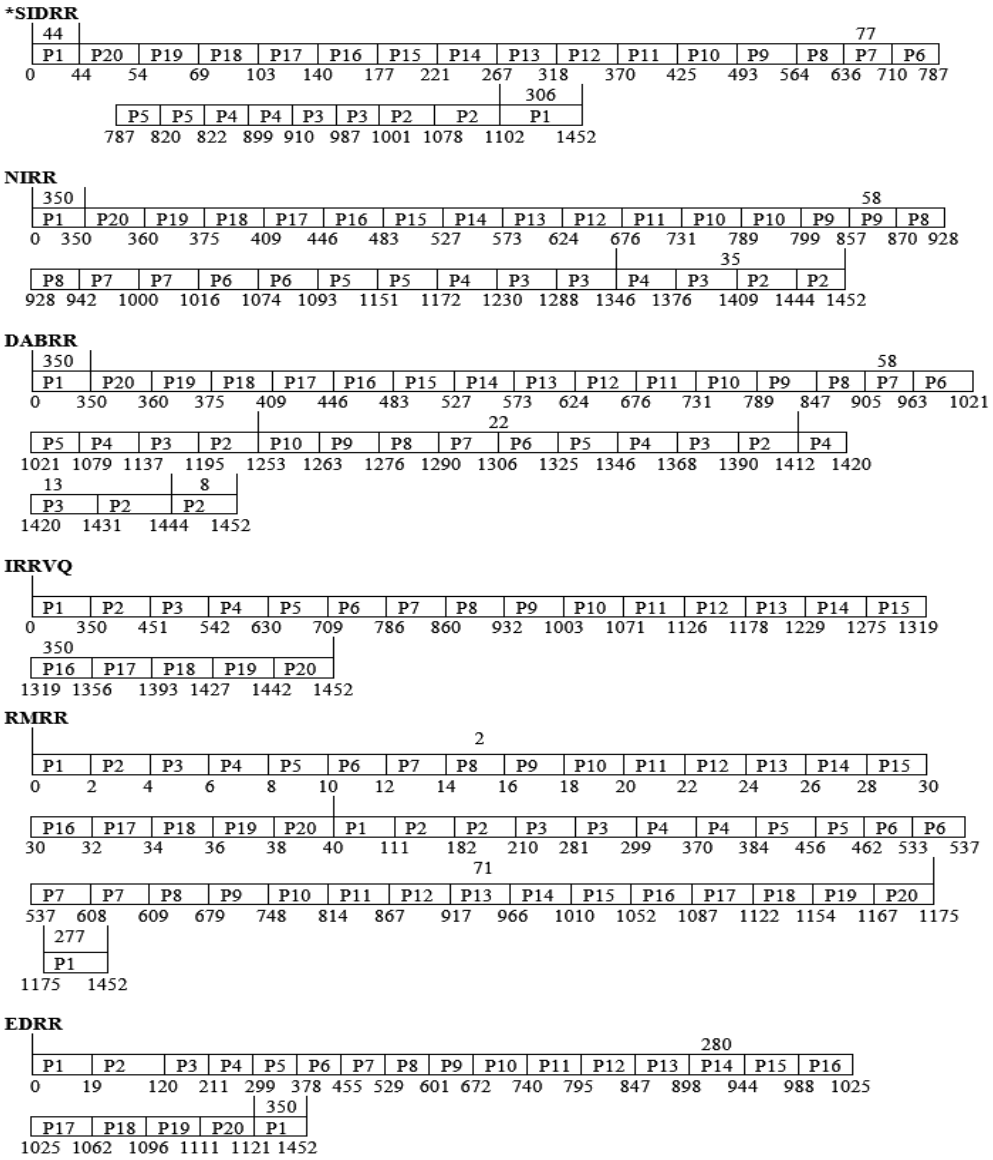


Figure 1. Flowchart of the proposed algorithm



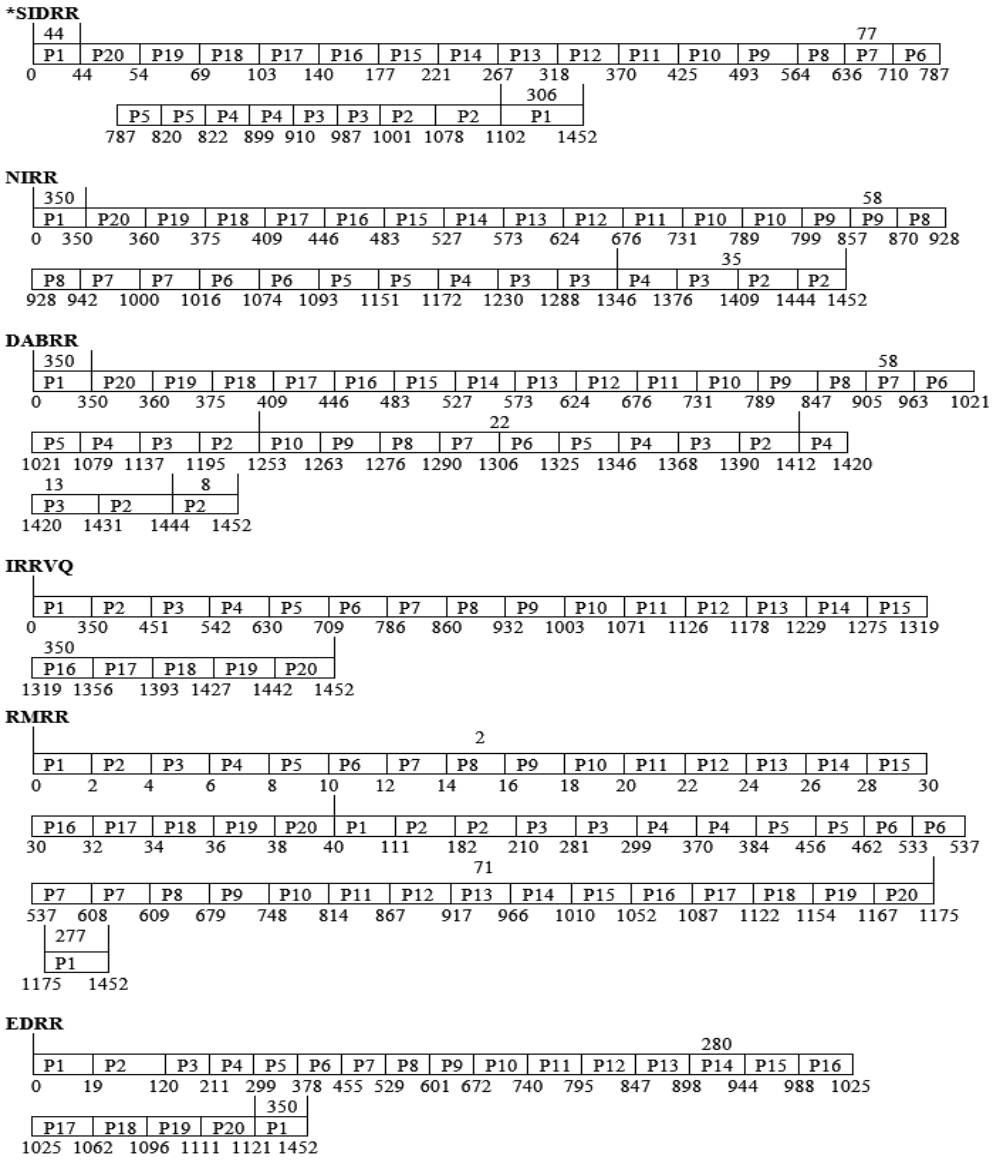


Figure 2. An illustrative Gantt chart of the selected algorithms and the proposed SIDRR

Table 1. Flow of experiments.

Zero arrival			Non-zero Arrival		
Random order	Ascending order	Descending order	Random order	Ascending order	Descending order

Table 2. Processes burst time in ascending order and zero arrival time.

Process ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Burst time	10	15	34	37	37	44	46	51	52	55	68	71	72	74	77	79	88	91	101	350
Arrival time	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3. Processes burst time in descending order and zero arrival time.

Process ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Burst time	350	101	91	88	79	77	74	72	71	68	55	52	51	46	44	37	37	34	15	10
Arrival time	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4. Processes burst time in random order and zero arrival time.

Process ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Burst time	51	77	44	10	79	34	88	68	72	74	15	55	91	37	71	101	350	52	37	46
Arrival time	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5. Processes burst time in ascending order and non-zero arrival time.

Process ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Burst time	10	15	34	37	37	44	46	51	52	55	68	71	72	74	77	79	88	91	101	350
Arrival time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 6. Processes burst time in descending order and non-zero arrival time.

Process ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Burst time	350	101	91	88	79	77	74	72	71	68	55	52	51	46	44	37	37	34	15	10
Arrival time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 7. Processes burst time in random order and non-zero arrival time.

Process ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Burst time	51	77	44	10	79	34	88	68	72	74	15	55	91	37	71	101	350	52	37	46
Arrival time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 8. Comparative result of zero arrival with ascending order processes

Measuring parameters	**IRRVQ (Manish & Rashid, 2014)**	**NIRR (Abdulrazaq et al., 2014)**	**DABRR (Dash et al., 2015)**	**RMRR (Kathuria et al., 2016)**	**EDRR (Farooq et al., (2017)**	***SIDRR (This Study.)**
Time quantum	10,5,19,3,7,2,5,1,3,13, 3,1,2,3,2,9,3,10,249	10,76,274	72,50,228	2,71,277	280,350	77,350
No of context switches	193	25	27	46	19	23
Average waiting time	793.55	425.45	501.05	763	425.45	425.45
Average turnaround time	866.15	498.05	573.65	835.6	498.05	498.05

Table 9. Comparative result of zero arrival with descending order processes

Measuring parameters	**IRRVQ (Manish & Rashid, 2014)**	**NIRR (Abdulrazaq et al., 2014)**	**DABRR (Dash et al., 2015)**	**RMRR (Kathuria et al., 2016)**	**EDRR (Farooq et al., (2017)**	***SIDRR (This Study.)**
Time quantum	10,5,19,3,7,2,5,1,3,13, 3,1,2,3,2,9,3,10,249	350,58,35	72,50,228	2,71,277	280,350	77,350
No of context switches	193	29	27	46	19	23
Average waiting time	793.55	711,55	501.05	444.45	676.55	425.45
Average turnaround time	866.15	784.15	573.65	517.05	749.15	498.05

Table 10 Comparative result of zero arrival with randomly ordered processes

Measuring parameters	**IRRVQ (Manish & Rashid, 2014)**	**NIRR (Abdulrazaq et al., 2014)**	**DABRR (Dash et al., 2015)**	**RMRR (Kathuria et al., 2016)**	**EDRR (Farooq et al., (2017)**	***SIDRR (This Study.)**
Time quantum	10,5,19,3,7,2,5,1,3,13, 3,1,2,3,2,9,3,10,249	51,74,276	72,50,228	2,71,277	280,350	77,350
No of context switches	193	25	27	46	19	23
Average waiting time	793.55	432.15	501.05	633.55	544.25	425.45
Average turnaround time	866.15	504.75	573.65	706.15	616.85	498.05

Table 11. Comparative result of non-zero arrival with ascending order processes

Measuring parameters	**IRRVQ (Manish & Rashid, 2014)**	**NIRR (Abdulrazaq et al., 2014)**	**DABRR (Dash et al., 2015)**	**RMRR (Kathuria et al., 2016)**	**EDRR (Farooq et al., (2017)**	***SIDRR (This Study.)**
Time quantum	10,5,19,3,7,2,5,1,3,13, 3,1,2,3,2,9,3,10,249	10,76,274	10,75,56,219	2,71,277	280,350	77,350
No of context switches	193	25	26	46	19	23
Average waiting time	784.05	415.95	472.2	434.95	415.95	415.95
Average turnaround time	856.65	488.55	544.8	507.55	488.55	488.55

Table 12 Comparative result of non-zero arrival with descending order processes

Measuring parameters	**IRRVQ (Manish & Rashid, 2014)**	**NIRR (Abdulrazaq et al., 2014)**	**DABRR (Dash et al., 2015)**	**RMRR (Kathuria et al., 2016)**	**EDRR (Farooq et al., (2017)**	***SIDRR (This Study.)**
Time quantum	350	350,58,35	350,58,35	2,71,277	280,350	77,350
No of context switches	19	29	32	46	20	24
Average waiting time	944.45	702.05	801.05	753.5	685.1	457.75
Average turnaround time	1017.05	774.65	873.65	826.1	757.7	530.35

Table 13. Comparative result of non-zero arrival with randomly ordered processes

Measuring parameters	**IRRVQ (Manish & Rashid, 2014)**	**NIRR (Abdulrazaq et al., 2014)**	**DABRR (Dash et al., 2015)**	**RMRR (Kathuria et al., 2016)**	**EDRR (Farooq et al., (2017)**	***SIDRR (This Study.)**
Time quantum	51,26,2,9,3,10,249	51,74,276	51,73,49,228	2,71,277	280, 350	77, 350
No of context switches	46	25	27	46	19	23
Average waiting time	749.4	422.65	499.3	564,4	534.75	422.65
Average turnaround time	822	495.25	571.9	637	607.35	495.25

Table 14. Number of Context Switches

	ZERO ARRIVAL				**NON-ZERO ARRIVAL**			
	Ascending	**Descending**	**Random**	**Total**	**Ascending**	**Descending**	**Random**	**Total**
IRRVQ	193	193	193	579	193	19	46	258
DABRR	27	27	27	81	26	32	27	85
RMRR	46	46	46	138	46	46	46	138
EDRR	19	19	19	57	19	20	19	58
NIRR	25	29	25	79	25	29	25	79
*SIDRR	23	23	23	69	23	24	23	70

Table 15. Average waiting time

	ZERO ARRIVAL				**NON-ZERO ARRIVAL**			
Algorithms	**Ascending**	**Descending**	**Random**	**Total**	**Ascending**	**Descending**	**Random**	**Total**
IRRVQ	793.55	793.55	793.55	2380.65	784.05	944.45	749.4	2477.9
DABRR	501.05	501.05	501.05	1503.15	472.2	801.05	499.3	1772.55
RMRR	763	444.45	633.55	1841	434.95	753.5	564.4	1752.85
EDRR	425.45	676.55	544.25	1646.25	415.95	685.1	534.75	1635.8
NIRR	425.45	711.55	432.15	1569.15	415.95	702.05	422.65	1540.65
*SIDRR	425.45	425.45	425.45	1276.35	415.95	457.75	422.65	1296.35

Table 16 Average turnaround time

	ZERO ARRIVAL				**NON-ZERO ARRIVAL**			
Algorithms	**Ascending**	**Descending**	**Random**	**Total**	**Ascending**	**Descending**	**Random**	**Total**
IRRVQ	866.15	488.55	866.15	2220.85	856.65	1017.05	822	2695.7
DABRR	573.65	573.65	573.65	1720.95	544.8	873.65	571.9	1990.35
RMRR	835.6	517.05	706.15	2058.8	507.55	826.1	637	1970.65
EDRR	498.05	749.15	616.85	1864.05	488.55	757.7	607.35	1853.6
NIRR	498.05	784.15	504.75	1786.95	488.55	774.65	495.25	1758.45
*SIDRR	498.05	498.05	498.05	1494.15	488.55	530.35	495.25	1514.15

Table 17. Percentage of average waiting time reduced by each algorithm

	IRRVQ	DABRR	RMRR	EDRR	NIRR	SIDRR
Grand total	4858.55	3275.7	3593.85	3282.05	3109.8	2572.7
Percentage reduced	0.00%	32.58%	26.03%	32.45%	35.99%	47.05%

Table 18. Percentage of average turnaround time reduced by each algorithm

	IRRVQ	DABRR	RMRR	EDRR	NIRR	SIDRR
Grand total	4916.55	3711.3	4029.45	3717.65	3545.4	3008.3
Percentage reduced	0.00%	24.51%	18.04%	24.39%	27.89%	38.81%