International Journal of Information Processing and Communication (IJIPC) Vol. 7 No. 2 [December, 2019], pp. 122-140

Online: ISSN 2645-2960; Print ISSN: 2141-3959

A Simplified Improved Dynamic Round Robin (SIDRR) CPU Scheduling Algorithm

[1]Omotehinwa, Temidayo Oluwatosin, [2]Azeez, Saeed Igbaoreto and [3]Olofintuyi, Sunday Samuel
[1,2,3]Department of Mathematical Sciences, Achievers University Owo, Nigeria
[1]oluomotehinwa@gmail.com, [2]sdbabatunde@gmail.com, [3]olofintuyi.sundaysamuel@gmail.com

Abstract

Background*: This paper proposed a Simplified Improved Dynamic Round Robin (SIDRR) algorithm that further improved on some existing Improvements on Round Robin CPU scheduling algorithms; Most of these improvements on Round Robin rely on arithmetic mean in selecting their Time Quantum (TQ). The arithmetic mean approach does not adequately represent the data.* ***Aim:*** *the aim of this study is to develop a simplified dynamic improved Round Robin CPU scheduling algorithm.* ***Method****: this study implemented five existing Round Robin scheduling algorithm using C++. The algorithms are; New Improved Round Robin (NIRR), Dynamic Average Burst Round Robin (DABRR), Improved Round Robin with Varying time Quantum (IRRVQ), Revamped Mean Round Robin (RMRR) and Efficient Dynamic Round Robin (EDRR). A new algorithm was also developed based on the Numeric Outlier Detection technique and geometric mean for dynamic time quantum determination. The proposed algorithm was compared with the five implemented using parameters such as average turnaround time, average waiting time and number of context switching.* ***Results****: the result of this study showed that the proposed algorithm performs better than the other five algorithms in terms of Average waiting time, average turnaround time & number of context switches. This study therefore, recommends the adoption of SIDRR for CPU scheduling and other emerging areas such as cloud computing resource allocation.*

Keywords: *CPU scheduling, SIDRR algorithm, Average Turnaround Time, Average Waiting Time, Context Switching*

1. Introduction

Central Processing Unit (CPU) Scheduling is a precise procedural set-plan of selecting which process gets access to the CPU for execution and once the current process leaves, what sequence others should follow to get the CPU. The basic idea is to keep the CPU busy as much as possible by executing a user process until it must wait for an event, and then switch to another process. Scheduling is an important practical problem in industry that is hard to tackle using traditional optimization methods, due to conflicting requirements, interrelated constraints, and high computational complexity (Gen & Cheng, 2000). Pati, Korde, and Dey, (2017) asserts that RR scheduling algorithm (one of the fundamental scheduling algorithms) is one of the most widely used CPU scheduling algorithms which are implemented in various software systems; used in various operating systems, such as Windows, Linux and so on. The LINUX operating system's heartbeat, which is the Kernel, is the engine room responsible for controlling the computer's resources and scheduling of users' jobs so that each one gets its fair share of system resources, including access to the CPU as well as the peripheral devices like disk and CD-ROM storage, printers and tape drives (Sobell, 1999). Adekunle, Ogunwobi, Jerry, Efuwape, Ebiesuwa, and Ainam (2014) in their study "A comparative study of scheduling algorithm for multiprogramming in real time system" indicates that there is actually no scheduling algorithm satisfying the condition of an ideal algorithm and concluded that further studies which improve current scheduling algorithms need to be done. This is validated in the study of Harshita, Subrata, and Ramya, (2017) that asserts that different CPU scheduling algorithms have peculiar properties favoring a particular class of processes over another. This paper proposes a new algorithm (SIDRR) that

Online: ISSN 2645-2960; Print ISSN: 2141-3959

further improved on some existing Improvements on Round Robin CPU scheduling algorithms; Most of these improvements on RR rely on arithmetic mean in selecting their Time Quantum (TQ). The arithmetic mean approach does not adequately represent the data. In this paper existing improvements on Round Robin scheduling algorithm proposed by the studies of Abdulrazaq, Saleh, & Junaidu, (2014) NIRR[1], Manish & Rashid, (2014) IRRVQ[2], Dash, Sahu, & Samantra, (2015) DABRR[3], Kathuria, Singh, Tiwar, & Prashant, (2016) RMRR[4], and Farooq, Shakoor, & Siddique, (2017) EDRR[5], were implemented in order to validate the results presented in each study and to ensure efficient comparative study of these algorithms with the (SIDRR) developed by this study.

2. Related Works

The core part of operating system just like any system, is scheduling. Some commonly used scheduling algorithms are: First in First Out (FIFO); Shortest Job First (SJF) without preemption; Preemptive SJF; Priority based without preemption; Preemptive Priority base; Round robin; Multilevel Feedback Queue.
The order in which a process is allocated and its duration is also determined by the algorithm (Jayavarthini, Chattopadhyay, Banerjee, & Dutta, 2017). Round Robin is particularly designed for time-sharing systems; each process has a small unit of CPU time (time quantum). This algorithm allows a process in the ready queue to run until its time quantum finishes, and then executes the next process in the ready queue. It is similar to First Come First Serve (FCFS) scheduling, but preemption is added to switch between processes. The ready queue is maintained as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1-time quantum. To implement the Round Robin scheduling, we keep the ready queue as a First-In-First-Out (FIFO) queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum, and dispatches the process ((Silberschatz, Gagne, & Galvin, 2009); (Mohammed, 2011); (Abdulrazaq, Salisu, Ahmad, & Saleh, 2014); (Kathuria, Singh, Tiwar, & Prashant, 2016)). Comparative analysis of some improved Round Robin Scheduling by Musa, Lasisi, and Gokir (2017) asserts that not all improvements are better than earlier works. The study confirmed DABRR algorithm (Dash, Sahu, & Samantra, 2015) as better than DTSRR algorithm (Muraleedharan, Antony, & Nandakumar, 2016) and RMRR algorithm (Kathuria, Singh, Tiwar, & Prashant, 2016) proposed later. In recent time different approaches have been used to increase the performance of CPU scheduling algorithms.

Manish and Rashid, (2014) Combines the Shortest Job First with conventional RR. The processes are sorted in ascending order. At every cycle, the shortest burst time is implemented as the time quantum.
Abdulrazaq, Saleh, and Junaidu, (2014) Combines First Come First Serve (FCFS) with SJN and conventional RR. TQ was computed using arithmetic mean.
Dash, Sahu, and Samantra, (2015) introduces Iterative computation of TQ using arithmetic mean at every CPU cycle. Similarly, Kathuria, Singh, Tiwar, and Prashant, (2016) utilizes arithmetic mean in computing the time quantum. The algorithm first assigns all burst times for TQ=2 while processes arrives in the Pre-ready queue. Remaining burst times are moved to ready queue and the mean is compute as TQ at every CPU cycle. the TQ is computed for remaining burst times. Any running process with remainder less than the TQ is reassigned to complete execution.

[1] New Improved Round Robin
[2] Improved Round Robin with Varying time Quantum
[3] Dynamic Average Burst Round Robin
[4] Revamped Mean Round Robin
[5] Efficient Dynamic Round Robin

Online: ISSN 2645-2960; Print ISSN: 2141-3959

The study by Singh and Dalal (2016) proposed an Improved Dynamic Round Robin (IDRR) for process scheduling. The time quantum was computed as

$$TQ = ceiling\left(\sqrt{(mean * \max(CPU\ burst\ time)) + (median * \min(CPU\ burst\ time))}\right) \quad (2.1)$$

The study give concession to SJF. Its TQ relies on both the arithmetic mean and median of the processes.

Farooq, Shakoor, and Siddique, (2017) also gives concession to Shortest Job First (SJF) by benchmarking the TQ and attempts to selectively run through processes to first execute the smaller tasks before returning CPU to the remaining larger burst times execution.

Neha and Jiyani (2018) in their study implemented a First Come First Serve (FCFS) with the conventional RR like Abdulrasaq et al. However, the mode of determining the time quantum was not clearly presented.

Zaidi and Shukla (2018) proposed a round robin with a variable quantum time in cloud computing environment. There work aimed at improving the machine efficiency and reducing the waiting time, turnaround time, context switches, response time and overall improving the machine performances. All the cloudlets are arranged in decreasing order of their burst time. The time quantum is derived by finding the square root of the product of the median and the highest burst time of the cloudlets. The method also adopts the normal round robin approaches in which the cloudlets are switched whenever they are unable to finish their task before the quantum time lapse. The result of the experiment was benchmarked with round robin algorithm considering only two metrics waiting time and turnaround time. The work performed better than the traditional round robin algorithm.

This study (Kumar, Kumar, Jain & Jain, 2018) presented an algorithm that uses round robin and shortest job first in order to reduce the turnaround time and waiting time. In their work, a fixed quantum time was used for all the processes. Whenever any process is unable to finish it task before the quantum time lapse, the process is placed back on the ready queue and the next process with shortest burst time takes up the CPU but if the process finishes it task before the quantum time lapse, the CPU can then take up another process with the shortest burst time for execution. The work was benchmarked with round robin, short job first using turnaround time, waiting time and context switches. The proposed algorithm performs better than round robin, shortest job first using the metrics aforementioned.

Fataniya and Patel (2018) presented another approach for scheduling algorithm of round robin using dynamic quantum time in cloud environment. In their methodology, the quantum time was obtained by taking the mean and median of all the burst time of the processes. The mean is considered by taking the average of all the burst time while the median was considered if the result was odd, them the middle number will be the median otherwise, if the result is even, then the median will be the mean of two central number. It can be observed from the review above that most of the methodologies adopted relied on arithmetic mean or shortest job first for time quantum determination. The obvious challenge with arithmetic mean is that the time quantum will always tend towards the mean if there is an outlying burst time. Also, shortest job first will lead to starvation for jobs with higher burst time.

3. Methodology

3.1 The proposed SIDRR Algorithm

//Gm: Geometric mean
//TQ: Time Quantum
//RQ: Ready Queue

Online: ISSN 2645-2960; Print ISSN: 2141-3959

//bt []: array of process' burst time
//bt: bt = | bt [] |
//n: number of process
//Pi: Process at ith index
//i, j: used as index on Queues
//Total: Total Burst Time
//Numeric outlier calculations: Median, 1st Quartile Q1
// 3rd Quartile Q3, Inter Quartile Range IQR
//IQR=Q3-Q1
//Inner fences: upper=Q3 + (IQR*1.5), lower=Q1 - (IQR*1.5)

[1] Start
[2] input n^{th} number of processes
[3] Total=1
[4] make a copy of RQ
[5] Arrange the processes in ascending order
[6] compute ***Numeric outlier parameters*** //Median, Q1, Q3, IQR, lower inner fences
[7] check for outlier in the Processes
[8] if (outlier > Q3) {
[9] Assign Q3 to TQ
[10] Allocate CPU to run time slice through the processes till n=0
[11] } else {
[12] If (n=1) {
[13] Arrange processes bt [] in ascending order
[14] end if
[15] }}
[16] end if
[17] move all process with bt>0 to RQ
[18] if (RQ=Null) {
[19] Compute AWT, ATAT, NCS
[20] DISPLAY RESULT
[21] TERMINATE
[22] } else {
[23] If (outlier>Q3) {
[24] Repeat step and till i=n
[25] Total*=burst time of process P_i provided bt[i]>0
[26] Compute TQ =$\prod_{i=1}^{n} Pi$
[27] } else {
[28] Compute TQ=$\prod_{i=1}^{n} Pi + Q1$
[29] }
[30] Allocate CPU
[31] If (bt<TQ/2||bt<TQ) {
[32] Re-allocate CPU
[33] If (RQ! = Null) {
[34] Repeat step 30 while n > 0
[35] i++;
[36] n--;

[37] }}

Online: ISSN 2645-2960; Print ISSN: 2141-3959

Processes are collected in the Ready Queue. A copy of the Ready Queue is made (solely for data set verification and computation of Numeric outlier parameters), sorted in ascending order, and the median, first Quartile, third Quartile as well as the Inter Quartile Range are computed. This is used to generate the Inner fences that verify the processes burst times that possibly skewed the data set. If there exist any outlier candidate in the data set and its value is greater than Q3, the time quantum is set to the 3rd Quartile and all the processes are allotted to the processor for the period of time quantum; subsequent burst time of processes greater than zero are moved to the Ready queue. Else, the first process is allocated to the processor for the period of its burst time; remaining processes are then rearranged in ascending order and moved to Ready Queue. While Ready queue is not empty, the geometric mean (equation 1) of all processes with burst time greater than zero are computed iteratively during execution cycle.

$$Gm = \prod_{i=1}^{rp} bt[i] \qquad (1)$$

If the outlier is less than Q3, $TQ = (Gm + Q1/2)$ the processes in the ready queue is allocated to the CPU, if the remaining time of the currently running process is less than half of the time quantum it is allow to complete its execution (computing the Geometric mean iteratively till all bt<0). Else, if the outlier is greater than Q3, Geometric mean is assigned to TQ, the processes in the ready queue is allocated to the CPU on TQ slot, if the remaining time of the running process is less than half of the time quantum, the process is re-allocated to complete its execution.

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

Online: ISSN 2645-2960; Print ISSN: 2141-3959

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

Online: ISSN 2645-2960; Print ISSN: 2141-3959

[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 get 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] BT_{max} = Maximum Burst Time;
[2] BT_i = Burst time of ith process

Online: ISSN 2645-2960; Print ISSN: 2141-3959

[3] QT = Quantum Time;
[4] N = Number of processes in ready queue;
[5] remainingProcesses = Remaining processes;
[6] i = 1;
[7] QT = 0.8 * BT_{max}
[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 = $BT_{max;}$
[18] i == 0;
[19] i ++;
[20] end

3.2 Evaluation Metrics

Definitions

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

$$TT_j = CT_j - AT_j \quad (2)$$

$$WT_j = TT_j - \beta t[i] \quad (3)$$

Hence, the Average Waiting Time (AWT) is:

$$AWT = \frac{1}{\gamma}\sum_{j=1}^{\gamma} WT_j \quad (4)$$

the Average Turnaround Time (ATT) is:

$$ATT = \frac{1}{\gamma}\sum_{j=1}^{\gamma} TT_j \quad (5)$$

The number of context switching is determined thus;

Online: ISSN 2645-2960; Print ISSN: 2141-3959

$$NCS = \sum_{i=1}^{\gamma} (\omega_i - 1) \quad (6)$$

Equations (2) to (6) are incorporated into the algorithm in 3.1. The algorithm was implemented using C++ programming language and the values for the metrics are generated. The selected algorithms IRRVQ, DABRR, RMRR, EDRR, NIRR were equally implemented in C++ with the values for the metrics generated for each algorithm.

4. Results and Discussion

4.1 Experimental Settings

First, we divided the problems into two types based on arrival time of processes (processes with zero arrival time and processes with non-zero arrival time). We further divided each into 3 more types based on the burst time of processes (in ascending order, descending order, & random order). We analyzed all the algorithms based on six situations. In each we have considered twenty processes with their arrival time and burst time.

4.1.1 Assumptions

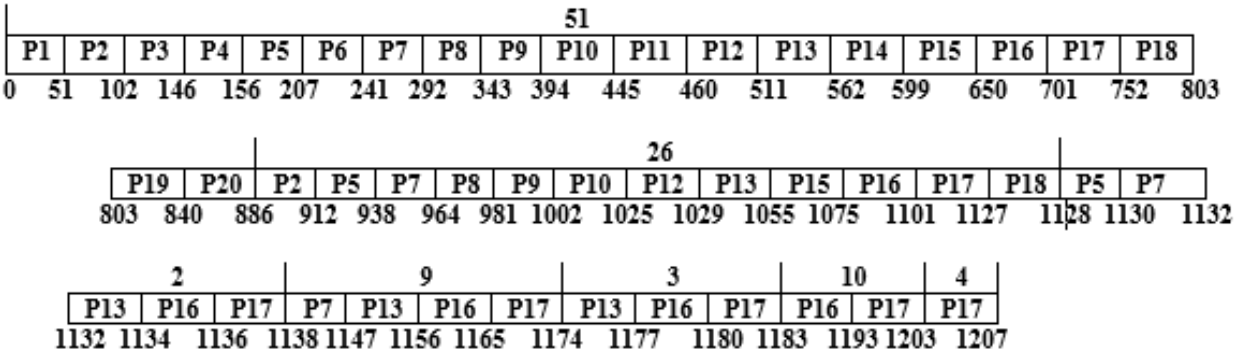
During analysis we have considered CPU bound processes only. In each test case 20 processes were analyzed in uni-processor environment. Corresponding burst time and arrival time of processes are known before execution. The context switch time of processes has been considered as zero. The time required for sorting the processes in ascending order also considered as zero.

Table 1: Process table (non-zero arrival)

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	105	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

The efficiency of each algorithm can be seen in the Gantt charts presented in Figure 1 to Figure 6.

IRRVQ:



Online: ISSN 2645-2960; Print ISSN: 2141-3959

Figure 1. Gantt chart representation of IRRVQ

NIRR:

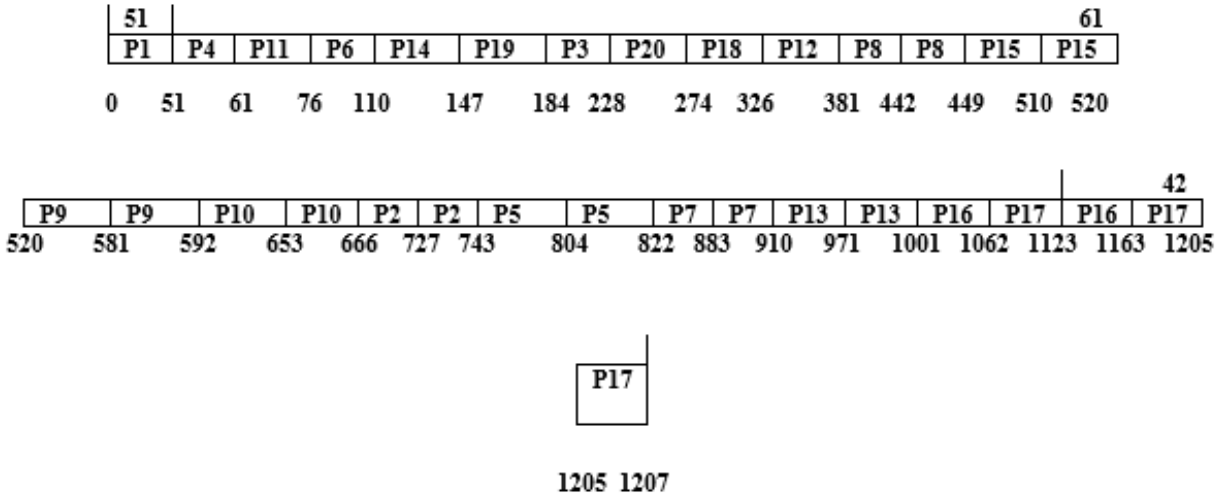


Figure 2. Gantt chart representation of NIRR

DABRR:

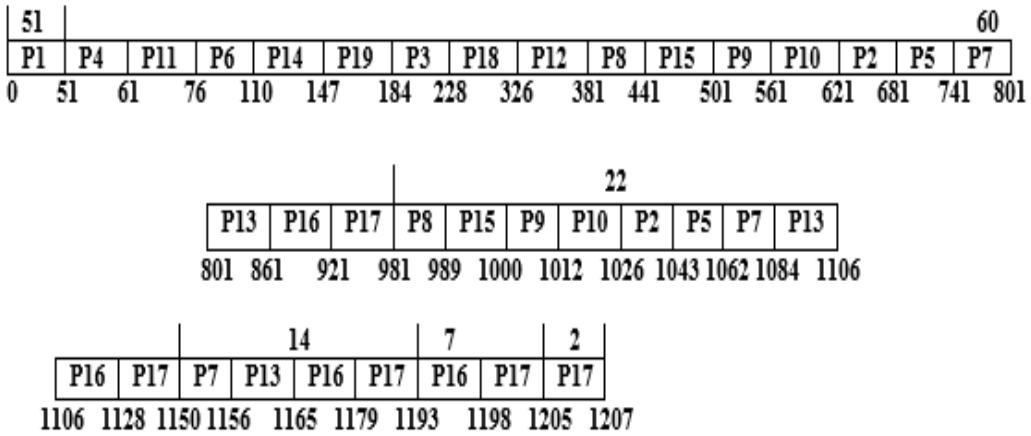


Figure 3. Gantt chart representation of DABRR algorithm

Online: ISSN 2645-2960; Print ISSN: 2141-3959

RMRR:

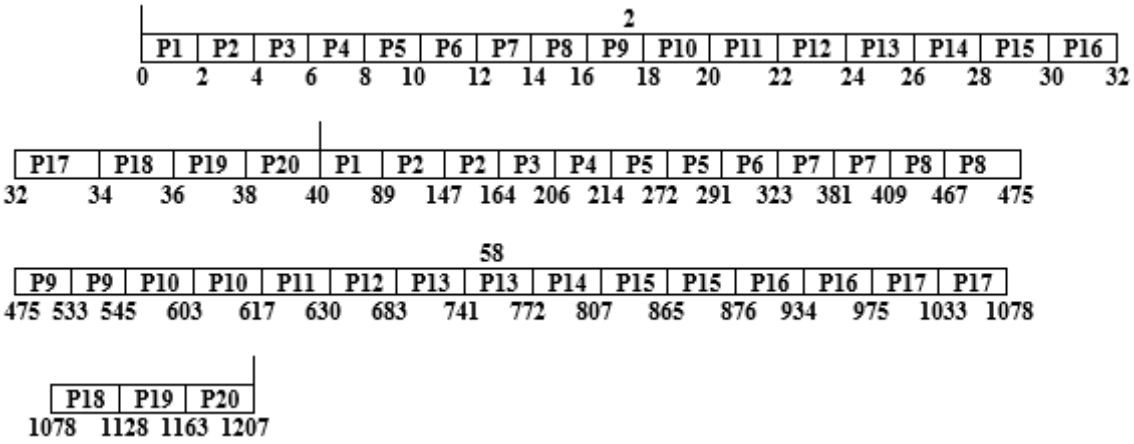


Figure 4. Gantt chart representation of RMRR algorithm

EDRR:

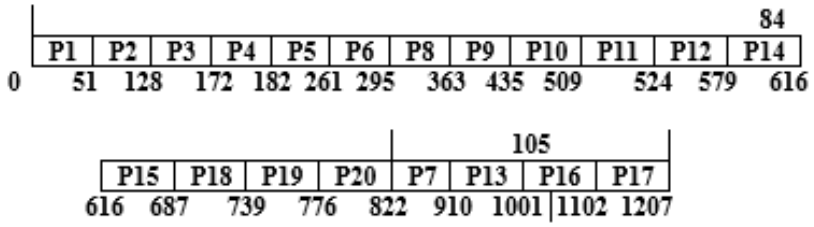


Figure 5. Gantt chart representation of EDRR algorithm

Proposed SIDRR:

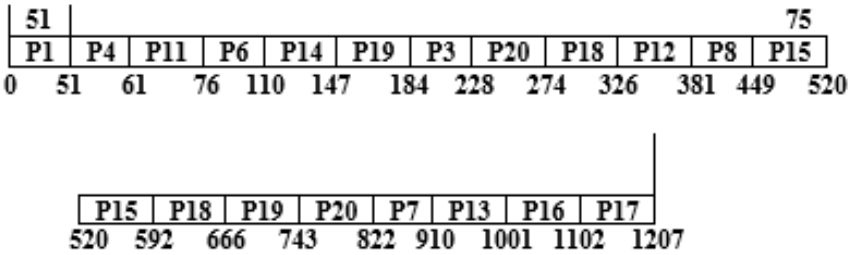


Figure 6. Gantt chart representation of SIDRR algorithm

Online: ISSN 2645-2960; Print ISSN: 2141-3959

Table 2. Comparison table between proposed algorithm and existing algorithms reviewed

	Existing algorithms and the proposed					
Measuring parameters	**IRRVQ**	**NIRR**	**DABRR**	**RMRR**	**EDRR**	***SIDRR**
Time quantum	51,26,2,9,3,10,4	51,61,42	51,60,22,14,7,2	2,58	84,105	51,75
No of context switches	46	30	36	49	19	19
Average waiting time	749.4	425.7	564.95	562.75	498.1	422.65
Average turnaround time	809.75	486.05	625.3	623.1	558.45	483

Table 2 presents the results of the execution of the settings in Table 1 on the algorithm developed by this study in comparison to existing algorithms selected. It can be observed from Table 2 that only EDRR equals the performance of *SIDRR as regards number of context switches.

Zero Arrival:

Given the same dataset as in Table 1, but with same arrival time; Table 3 represents the detail results of the selected algorithms under study and the proposed.

Table 3. Comparative result of zero arrival with randomly ordered processes

	Existing algorithms and the proposed					
Measuring parameter	**IRRVQ**	**NIRR**	**DABRR**	**RMRR**	**EDRR**	***SIDRR**
Time quantum	10,5,19,3,7,2,5,1,3,13,3,1,2,3,2,9,3,10,4	51,61,42	60,22,14,7,2	2,58	84,105	51,75
No of context switches	193	30	36	49	19	19
Average waiting time	793.55	435.2	567.75	612.4	507.6	432.15
Average turnaround time	853.9	495.55	628.1	672.75	567.95	492.5

Ascending order, Zero Arrival

Considering the same dataset earlier, but in ascending order in Table 4 with same arrival time;

Online: ISSN 2645-2960; Print ISSN: 2141-3959

Table 4. Process table (ascending order)

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	105
Arrival time	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The result of implemented algorithms is given in Table 5;

Table 5. Comparative result of zero arrival with ascending order processes

	Existing algorithms and the proposed					
Measuring parameter	**IRRVQ**	**NIRR**	**DABRR**	**RMRR**	**EDRR**	***SIDRR**
Time quantum	10,5,19,3,7,2,5,1,3,13,3,1,2,3,2,9,3,10,4	10,63,40	60,22,14,7,2	2,58	84,105	10,80
No of context switches	193	30	36	49	19	19
Average waiting time	793.55	428.6	567.75	740.2	425.45	425.45
Average turnaround time	853.9	488.95	628.1	800.55	485.8	485.8

Table 6 presents the result of descending order without outlier, if the dataset is arranged in descending order and non-zero.

Table 6. Comparative result of non-zero arrival with descending order processes

	Existing algorithms and the proposed					
Measuring parameters	**IRRVQ**	**NIRR**	**DABRR**	**RMRR**	**EDRR**	***SIDRR**
Time quantum	10,5,19,3,7,2,5,1,3,13,3,1,2,3,2,9,3,10,4	105,58,35	60,22,14,7,2	2,58	84,105	105,73
No of context switches	193	29	36	49	19	19
Average waiting time	793.55	478.8	567.75	444.45	577.6	470.1
Average turnaround time	853.9	539.15	628.1	504.8	637.95	530.45

Online: ISSN 2645-2960; Print ISSN: 2141-3959

The next scenario is the descending order without outlier, if the dataset is arranged in descending order according to Table 7.

Descending order, zero arrival time

Table 7. Process 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	105	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

The result of descending order, zero arrival time and descending order non-zero arrival time are presented in Tables 8 and 9 respectively.

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

	Existing algorithms and the proposed					
Measuring parameters	**IRRVQ**	**NIRR**	**DABRR**	**RMRR**	**EDRR**	***SIDRR**
Time quantum	10,5,19,3,7,2,5,1,3,13,3,1,2,3,2,9,3,10,4	105,58,35	60,22,14,7,2	2,58	84,105	105,73
No of context switches	193	29	36	49	19	19
Average waiting time	793.55	478.8	567.75	444.45	577.6	470.1
Average turnaround time	853.9	539.15	628.1	504.8	637.95	530.45

Also, the result of the non-zero arrival is given in Table 9;

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

	Existing algorithms and the proposed					
Measuring parameters	**IRRVQ**	**NIRR**	**DABRR**	**RMRR**	**EDRR**	***SIDRR**
Time quantum	105	105,58,35	105,58,22,13,8	2,58	84,105	105,73
No of context switches	19	29	32	49	19	19
Average waiting time	711.7	469.3	568.3	730.7	568.1	460.6
Average turnaround	772.05	529.65	628.65	791.05	628.45	520.95

Online: ISSN 2645-2960; Print ISSN: 2141-3959

time

Comparative analysis

This section provides the comparative analysis of six algorithms on the basis of their resulted number of context switches, average waiting time, and average turnaround time. Tables 10 - 12 shows the performance analysis of six algorithms by summarizing the number of context switches, average waiting time, and average turnaround time resulted from case I (Zero arrival) and II (Non-zero arrival).

It can be observed in Table 10 that for IRRQ, NIRR, DABRR, and RMRR the number of preemptions is higher when compared to the proposed algorithm SIDRR. This is because the time quantum is either determined by arithmetic mean of burst times or Shortest Job First. EDRR's time quantum is dynamically determined based on burst time of processes. The high time quantum also impacted the average waiting time and average turnaround time as can be observed in Tables 11 and 12.

Table 10. 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	36	36	36	108	36	32	36	104
RMRR	49	49	49	147	49	49	49	147
EDRR	19	19	19	57	19	19	19	57
NIRR	30	29	30	89	30	29	30	89
*SIDRR	19	19	19	57	19	19	19	57

Table 11. Average waiting time

	ZERO ARRIVAL				NON-ZERO ARRIVAL			
	Ascending	**Descending**	**Random**	**Total**	**Ascending**	**Descending**	**Random**	**Total**
IRRVQ	793.55	793.55	793.55	2380.65	784.05	711.7	749.4	2245.15
DABRR	567.75	567.75	567.75	1342.6	564.1	568.3	564.95	1697.35
RMRR	444.45	740.2	612.4	1703.25	434.95	730.7	562.75	1728.4
EDRR	577.6	425.45	507.6	1797.05	145.95	568.1	498.1	1212.15
NIRR	478.8	428.6	435.2	1510.65	419.1	469.3	425.7	1314.1
*SIDRR	470.1	425.45	432.15	1327.7	145.95	460.6	422.65	1029.2

Table 12. Average turnaround time

	ZERO ARRIVAL				NON-ZERO ARRIVAL			
	Ascending	**Descending**	**Random**	**Total**	**Ascending**	**Descending**	**Random**	**Total**
IRRVQ	853.9	853.9	853.9	2561.7	844.4	772.05	809.75	2426.2
DABRR	628.1	628.1	628.1	1884.3	624.45	628.65	625.3	1878.4
RMRR	800.55	504.8	672.75	1978.1	495.3	791.05	623.1	1909.45
EDRR	485.8	637.95	567.95	1691.7	476.3	628.45	558.45	1663.2
NIRR	488.95	539.1	495.55	1523.65	479.45	529.65	486.05	1495.15
*SIDRR	485.8	530.45	492.5	1508.75	476.3	520.95	483	1480.25

The corresponding charts for the comparison tables is given below,

Online: ISSN 2645-2960; Print ISSN: 2141-3959

Tables 10 – 12 show that SIDRR, the algorithm developed by this study performed better than the other five algorithms implemented on same processes with zero and non-zero arrival time. It can also be observed that only EDRR has an equal performance in terms of number of context switches.

Considering the IRRVQ as the baseline improvement over the conventional RR, the result analysis affirms SIDRR developed by this study is an improvement over some of the algorithms in literature.

Figures 7 and 8 depicts the percentage of average waiting time saved by each algorithm than Improved Round Robin Varying time Quantum algorithm. Proposed algorithm SIDRR saves 49.05% waiting time as compared with IRRVQ, DABRR, RMRR, EDRR and NIRR.

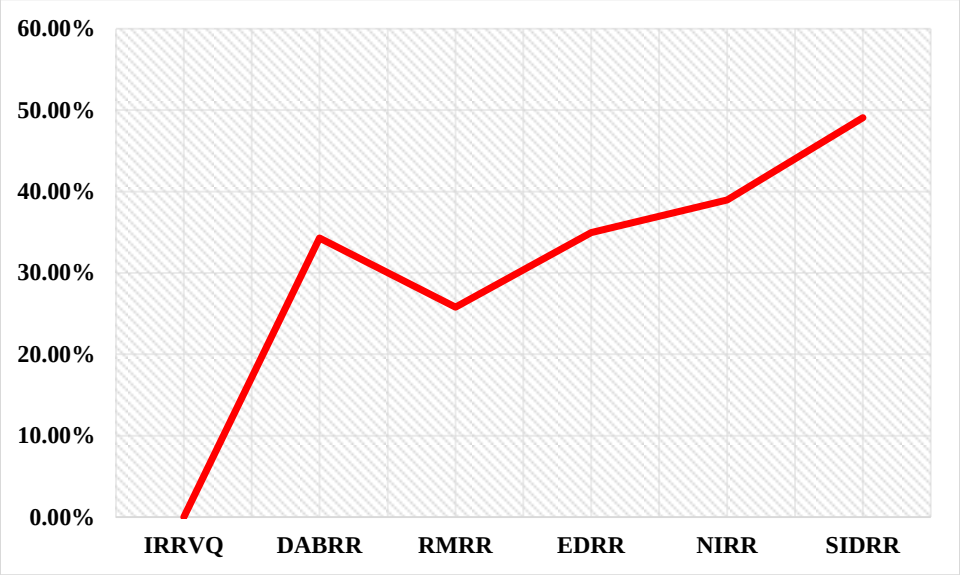


Figure 7. Graph showing percentage reduction in average waiting time by each algorithm

Online: ISSN 2645-2960; Print ISSN: 2141-3959

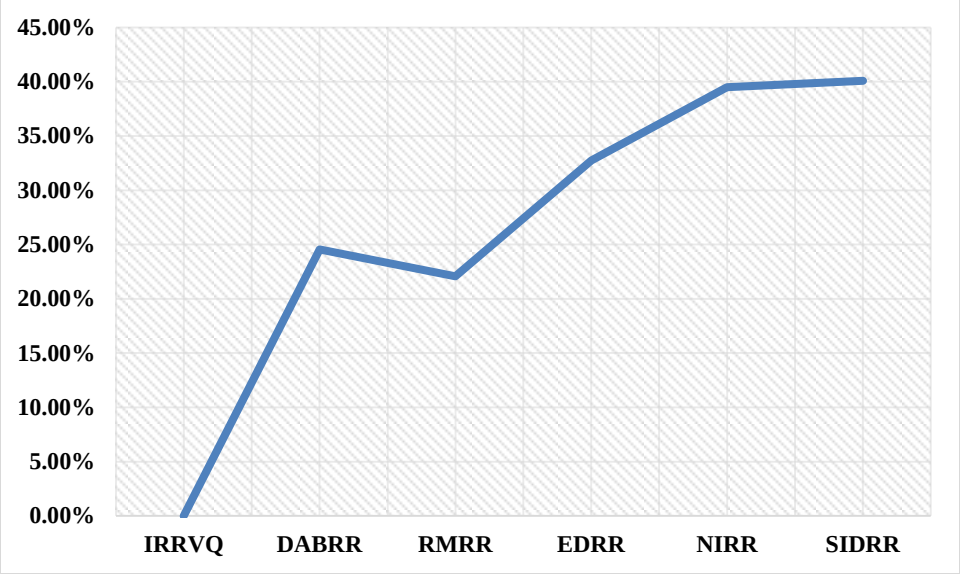


Figure 8. Graph showing percentage reduction in average turnaround time by each algorithm

Conclusions

This study focused at developing an improved dynamic RR algorithm that is not dependent on arithmetic mean in determining its time quantum such that when an outlier is detected within the burst time of processes the TQ is dynamically determined. The study selected and studied five existing improvements on Round Robin Scheduling algorithms. The algorithms are; New Improved Round Robin (NIRR), Dynamic Average Burst Round Robin (DABRR), Improved Round Robin with Varying time Quantum (IRRVQ), Revamped Mean Round Robin (RMRR) and Efficient Dynamic Round Robin (EDRR). A new algorithm was also developed based on the Numeric Outlier Detection technique and geometric mean for dynamic time quantum determination. The five selected algorithms and the proposed algorithm were implemented in C++ programming language. The algorithms were evaluated based on average waiting time, average turnaround time, and number of context switching. The results presented in this paper demonstrated improved performance in comparison to NIRR, DABRR, IRRVQ, RMRR and EDRR in terms of average waiting time and average turnaround time. The performance in terms of number of context switches is the same for only EDRR and the proposed algorithm. This study therefore, recommends the adoption of SIDRR for CPU scheduling and other emerging areas such as cloud computing resource allocation.

Reference

Abdulrazaq, A., Saleh, E. A., & Junaidu, S. B. (2014). A New Improved Round Robin (NIRR) CPU Scheduling Algorithm. *International Journal of Computer Applications. 90*(4), 27-33, 0975-8887

Abdulrazaq, A., Salisu, A., Ahmad, M. M., & Saleh, E. A. (2014). An Additional Improvement in Round Robin (AAIRR) CPU Scheduling Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering, 4*(2), 601-610.

Online: ISSN 2645-2960; Print ISSN: 2141-3959

Adekunle, Y. A., Ogunwobi, Z. O., Jerry, S. A., Efuwape, B. T., Ebiesuwa, S., & Ainam, J. (2014). A comparative Study of Scheduling Algorithms for Multiprogramming in Real-Time Systems. *International Journal of Innovation and Scientific Research, 12*(1), 180-185. Retrieved February 13, 2019, from http://www.ijisr.issr-journals.org/abstract.php?article=IJISR-14-272-17

Dalal, S., & Singh V. (2016). Improved Dynamic Round Robin Process Scheduling. *International Journal of Advanced Technology in Engineering and Science (IJATES), 4*(1), 128-132

Dash, A. R., Sahu, S. k., & 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), 5*(1), 7-26.

Farooq, M. U., Shakoor, A., & Siddique, A. (2017). An Efficient Dynamic Round Robin Algorithm fo CPU Scheduling. *IEEE International Conference on Communication, Computing and Digital Systems* (pp. 244-248). IEEE. doi:10.1109/C-CODE.2017.7918936

Fataniya, B. & Patel, M. (2018). Dynamic Time Quantum Approach to Improve Round Robin Scheduling Algorithm in Cloud Environment. *International Journal of Scientific Research in Science, Engineering and Technology*, *4*(4), 963-969

Gen, M., & Cheng, R. (2000). *Genetic Algorithm and Engineering Optimization.* New York: JohnWiley & sons, inc.

Jayavarthini, C., Chattopadhyay, A., Banerjee, P., & Dutta, S. (2017). Performance analysis of CPU scheduling algorithms with DFRRS algorithm. *ARPN Journal of Engineering and Applied Sciences, 12*(12) 3757-3761.

Kumar, S., Kumar, G., Jain, K., & Jain, A. (2018). An approach to reduce turn around time and waiting time by the selection of round robin and shortest job first algorithm. *International Journal of Engineering & Technology (IJET) 7*(2.8) 667-672.

Kathuria, S., Singh, P. P., Tiwar, P. i., & Prashant. (2016). A Revamped Mean Round Robin (RMRR) CPU Scheduling Algorithm. *International Journal of Innovative Research in Computer and Communication Engineering*, *4*(4), 6684-6691.

Manish, K. M., & Rashid, F. (2014). An Improved Round Robin CPU Scheduling Algorithm with Varying Time Quantum. *International Journal of Computer Science, Engineering and Applications (IJCSEA), 4*(4), 1-8.

Mohammed, A. H. (2011). A selective Quantum of Time for Round Robin Algorithm to Increase CPU Utilization. *International Journal of Computer Information Systems, 3*(2), 54-59.

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

Muraleedharan, A., Antony, N., & Nandakumar, R. (2016). Dynamic Time Slice Round Robin Schedulling Algorithms with Unknown Burst Time. *Indian Journal of Science and Technology, 9*(8), 1-6.

Neha & Jiyani, A. (2018). An Improved Round Robin CPU Scheduling Algorithm. *Iconic Research and Engineering Journals, 1(9),* 82-86

Online: ISSN 2645-2960; Print ISSN: 2141-3959

Pati, M. S., Korde, P., & Dey, P. (2017). An Advanced Approach to Traditional Round Robin CPU Scheduling Algorithm to Prioritize Processes with Residual. *IOP Conference Series: Materials Science and Engineering. 263*, 042001 pp. 1-9. IOP Publishing. doi:10.1088/1757-899X/263/4/042001

Shubham, M., Swati, S., & Ramandeep, K. (2016). Enhance Round Robin Technique for Task Scheduling in Cloud Computing Enviroment. *International Journal of Engineering Research & Technology (IJERT), 5*(10), 525-529.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2009). *Operating System Concepts* (8th ed.). New Jersey, United States of America: John Wiley & Sons, Inc.

Sobell, M. G. (1999). *A practical guide to Linux.* USA: Addison-Wesley Longman, Inc.

Zaidi, T & Shukla, S (2018). Variable time Quantum Based Round Robin Policy for Cloud Computing Environment. *International Journal of Computer Sciences and Engineering (IJCSE), 6*(5), 1075-1081.