Open Access
Full Length Article

ORIGINAL ARTICLE

Randomized Dynamic Quantum CPU Scheduling Algorithm

Rabia Riaz[1], Sobia Hassan Kazmi[2], Zaki Hassan Kazmi[3], Saeed Arif Shah[4]

[1,2,3,4] Department of CS & IT, University of Azad Jammu and Kashmir, Muzaffarabad, 13100, Pakistan

ABSTRACT

Scheduling plays an essential role in multitasking, multiprocessing and real time operating systems. The major objective of scheduling is to enhance system performance. Various preemptive and non-preemptive scheduling techniques have been developed; however, often contradictory results have left designer skeptical about the efficacy of techniques and there exists no clear consensus about robustness of these scheduling technique. Round robin is considered as most widely adopted algorithm having advantages that include fairness and simplicity, but constant time quantum may result in worst turnaround time and waiting time. In this research work, a new CPU scheduling algorithm, Randomized Dynamic Quantum (RDQ), based on round robin scheduling has been proposed that is based on the idea of using randomly generated dynamic time quantum. Simulation results indicate that RDQ improves turnaround time and waiting time as compared to existing schemes.

.**Keywords:** CPU scheduling, optimization, round robin, dynamic quantum

Author`s Contribution
[1,2,3,4,]*Manuscript writing, Data analysis, interpretation, Conception, synthesis, planning of research, Interpretation and discussion, Data Collection*

Address of Correspondence
Rabia Riaz
Email: rabiaiqbal18@gmail.com

Article info.
Received: Feb 9, 2018
Accepted: Dec 11, 2018
Published: Dec 30, 2018

Cite this article: *Riaz R, Kazmi SH, Kazmi ZH, Shah SA. Randomized Dynamic Quantum CUP Scheduling Algorithm. J. Inf. commun. technol. robot. appl.2018; 9(2):19-27*

Funding Source: *Nil*
Conflict of Interest: *Nil*

INTRODUCTION

Scheduling plays an integral role in order to optimize performance of CPU in handling different tasks given by the user. System where single processor is active can only run one process at a time; other processes wait for allocation of the processor until the active process finishes its execution. The goal of multiprogramming is to maximize the utilization of processor by keeping it busy all the time in executing some processes. The scheduling is an important function of the operating system that makes possibility of resource sharing. All the system resources are scheduled before its utilization. CPU is considered as primary resource of computer, thus in the operating system design its scheduling plays an important role in increasing the performance of the system.

1. CPU Scheduling Levels

The CPU scheduling is broken down into three separate phases:

Long Term Scheduler

Long term scheduler is also known as job or admission scheduler. Its purpose is to identify the job or process that will be allowed to compete for the resources of the system. The main objective of the admission scheduler is to allocate appropriate job to mid-term scheduler. It is not invoked frequently but it control the multiprogramming degree. It may be passive on certain systems like time sharing systems. Its purpose is to do careful selection of CPU bound as well as I/O bound processes.

Medium Term Scheduler

It is also called swapper because it is part of a swapping function. The medium-term or intermediate scheduler removes job that is in main memory storage area and places it in secondary memory like disk, or else removes job from disk and places back to main memory. This procedure commonly called swapping in of process and swapping out of process. It is commonly present in time-sharing systems.

Short Term Scheduler

It is also known as CPU scheduler. It is a short-term scheduler decision to assign the CPU to any one of the processes for execution. At this level the scheduling goes through by dispatcher that allows the specific process to acquire CPU for further execution. The CPU scheduler is invoked frequently. It should be fast. It dissipates certain time slice of processor during scheduling process. Whenever any event like calls regarding to operating system, clock interrupts and I/O interrupts takes place, it preempts the current process in block state.

2. CPU Scheduling Criteria

The CPU scheduling algorithms can be compared on the basis of different criterion and this can lead to determine the best among several algorithms. **CPU Consumption:** All the time CPU must be busy in doing some sort of useful work. It is the percentage of the time that the CPU is busy in executing a process or job. The CPU utilization should be maximized.

Throughput: It refers to the total number of processes completed in a unit time. The amount of work done by the CPU should be maximized.

Turnaround Time: It is the total estimated time between process compliance and its completion by the system. It should be minimized.

Response Time: As compare to turnaround time the response time is the estimated time from the request submission unless the initial response is generated. It should be minimized.

Waiting Time: It refers to the total time that a process spent in waiting state in the ready queue. It should be minimized.

3. CPU Scheduling Algorithms

CPU scheduling is a technique used for allocation of processor to the processes in a strategical manner according to particular criteria. Different techniques are used to select the process to which CPU control is given [1]-[5]. CPU scheduling purpose is to maximize CPU utilization. In order to maximize CPU utilization one process should be running on CPU all the time. To achieve this level of accuracy we have different scheduling techniques. The four well known CPU scheduling algorithms are discussed in this section. Table 1 explains the symbols used in algorithms.

Table 1. Symbols Description used in Algorithms

Description	Symbol
Process i	P[*i*]
Total Number of active Processes	N
Burst Time of Process i	B_T *[i]*
Waiting time of Process i	W_T *[i]*
Turn Around Time	TA_T
Remaining Burst Time	Rem_ B_T
sQuantum	Qtm_T
Average Waiting Time	Avg W_T
Average Turnaround Time	Avg TA_T

First-Come First-Served (FCFS) Scheduling

It is one of the simplest techniques. In FCFS a process whose request comes first in queue gets the control of CPU. FCFS algorithm is non-preemptive. The process holds the CPU until its execution completes or waits of I/O events. It could easily be implemented by using First In First out (FIFO) queue. When a process finishes its execution the processor is taken away and allotted to next ready process in the queue [6]. It is not appropriated for time sharing systems [7]. In FCFS technique process which has long burst time will execute first. Then, next process with longest burst time will be fetched. Process with smaller burst time will have to wait until process with long burst time completes its execution. This situation is called convoy effect. FCFS is not used now a day because its performance is not good. It gives bad performance, lower throughput, lower average time and much longer turnaround time. Scheme is explained in Algorithm 1.

ALGORITHM 1	FCFS Scheduling
1	Input the processes along with their burst time (B_T);
2	**Waiting time (W_T)** for all processes;
	i. As first process that comes need not to wait so

	waiting time for process 1 will be 0 i.e. $W_T[1] = 0$;
	ii. Waiting time for all other processes can be calculated P[i] waiting time $W_T [i]= B_T [i-1]+ W_T [i-1]$;
2	**Turnaround time**, $TA_T = B_T + W_T$ for all processes;
3	**Average waiting time** = $Avg\ W_T = \frac{\sum_{i=0}^{N} W_T[i]}{N}$;
4	**Average turnaround time** = $Avg\ TA_T = \frac{\sum_{i=0}^{N} TA_T[i]}{N}$;

Shortest Job First (SJF) Scheduling

The technique in which the job with shortest burst time is executed first and then so on, but if two or more jobs have same burst time then FCFS technique is used in that scenario. SJF technique could be either one preemptive or non-preemptive. The choice is made when at the ready queue the new process is arrived while the prior process is running preemptive, it is called shortest time remaining first (STRF) [7] [8]. SJF scheduling is optimal in most scenarios because it has minimum average waiting time and minimum average turnaround time [9]. But in SJF there is an overhead of starvation for the jobs with higher burst time [10]. Scheme is explained in Algorithm 2.

ALGORITHM 2 **SJF Scheduling**
1. Sort all the processes in increasing order according to burst time;
2. Then simply, apply FCFS;

Round Robin (RR) Scheduling

It is most widely used technique in operating systems and is specially designed for time sharing systems [7]. A small amount of time is associated with each job called time quantum or time slice. All ready processes are placed in queue. The scheduler allots CPU to the initial process which is at the head of queue for specific time quantum and new coming processes are added at tail with similar time quantum associated. Here two situations could occur:

- If process has completed its execution before ending of it time slice it will release the CPU and then CPU is allocated to next job which is at the head of ready queue.
- If the CPU burst of executing process is higher than time quantum then processor is forcefully taken away from that process and allocated to next selected process. The preempted process is added at the tail (end) of the queue, this procedure is called context switching.

The performance of RR technique depends on the quantum size [6] [11]. If time quantum is too short then due, to a lot of context switching efficiency of CPU decreases. But if time quantum is too long it causes poor response time and estimates FCFS [12]. Turnaround time of round robin algorithm also depends upon time quantum. In RR algorithm average waiting time is high. Scheme is explained in Algorithm 3.

ALGORITHM 3	**RR Scheduling**
1	Create an array Rem_B_T [ ] to keep track of remaining burst time of processes. $Rem\_ B_T[i]=B_T[i]$;
2	Create another array W_T [ ] to store waiting times of processes. Initialize this array as $W_T[i]=0$;
3	Initialize time: t = 0;
4	While all processes are not done {
	For each process i do following {
	If $Rem\_ B_T[i] > Qtm_T$
	t = t + QtmT
	$Rem\_ B_T[i] = Rem\_ B_T[i] - Qtm_T$;
	Else
	$t = t + Rem\_ B_T[i]$;
	$W_T[i]= t - B_T[i]$; $Rem\_ B_T[i] = 0$;
	Decrement N;}}
5	Compute turnaround time: $TA_T[i] = W_T[i] + B_T[i]$;

Priority Scheduling

In this technique, priority is associated with each process that is utilized to allocate the control of processor [10]. Process with highest priority will get CPU control first and after that next process with highest priority will take control. When two processes come with similar priority then FCFS technique is implemented.

Priority scheduling could be defined as preemptive or non-preemptive. In preemptive priority process scheduling, a process is currently executing and another process comes with higher priority than CPU is taken forcefully from executing process and given to higher priority process. While in non-preemptive priority scheduling executing process is not preempted until or unless it completes its execution. The main problem that occurs with priority scheduling algorithm is indefinite blocking or starvation when lower priority process has to wait longer for their execution. This problem could be resolved by a technique called aging. Scheme is explained in Algorithm 4.

ALGORITHM 4	**Priority Scheduling**
1	First input the processes with their burst time and priority;
2	Sort the processes, burst time and priority according to the priority;
3	Now simply apply FCFS algorithm;

In this paper, is organized in following sequence. In Section 1, the basic concept and the importance of CPU scheduling are discussed. It has been tried to explore all attribute of scheduling to help further researchers for more creativity in this field. In Section 2, different concepts and methodologies of previous round robin scheduling algorithms are discussed. Moreover the loop holes in previous algorithms have also been identified, which becomes base for new developed algorithm. In Section 3, the methodological details of proposed algorithm are discussed. In Section 4 on the basis of simulation and generated results the performance of new developed algorithm is compared with the performance of some existing scheduling algorithms. At the end of the paper, the summary and recommendations are presented in Section 5.

VARIANTS OF ROUND ROBIN ALGORITHM

The round robin CPU scheduling technique is widely used in most operating system, even with its overhead due to static predefined time quantum [7]. In order to fill this gap the researchers have developed various techniques; few of them are discussed in this section.

Burst Round Robin as a Proportional-share Scheduling Algorithm

Helmy and Dekdouk [13] proposed a different weighting scheme for existing round robin algorithm. The proposed scheme is an effort to combine overhead of low scheduling in round robin and favor smaller processes. Large quantum means that the process have associated higher weights; more time will be given to smaller jobs so they will be removed earlier from the ready queue.

Self-Adjustment Round Robin (SARR) Algorithm

Matarneh [11] proposed an algorithm that is based on the idea that an optimum time quantum value can be calculated by considering the median parameter of the burst times of waiting processes present in the ready queue, only if this median value is not more than 25ms. In such situation the time quantum value must be readjusted to 25ms in order to reduce CPU overhead due to high context switching time.

Shortest Remaining Burst Round Robin (SRBRR) Scheduling Algorithm

Mohanty et al [14] proposed an algorithm that is the combination of shortest job first and round robin algorithms. It gives better results as compare to existing round robin algorithm.

Priority Based Dynamic Round Robin (PBDRR) Algorithm

Mohanty et al [15] proposed another algorithm in order to make better performance of scheduling algorithms. The proposed algorithm is designed by integrating the existing priority based technique and round robin technique.

Weighted mean Priority Based Scheduling Algorithm

Behera [16] proposed a new process scheduling algorithm that is based on the idea of dynamic time quantum and weighted mean.

Modified mean-Deviation Round Robin (MMDRR) Scheduling Algorithm

Behera [17] proposed a new algorithm that is based on the idea of combining two schemes such as mean-deviation time-quantum and increasing order burst time is executed in RR algorithm.

Time Quantum Based Improved Scheduling Algorithm (TQBISA)

Kishore and Goyal [18] designed an algorithm that is integration of shortest job first (SJF) and round robin (RR) algorithm.

▪ **AN Algorithm**

Noon et al [19] proposed an algorithm named as AN algorithm, to overcome the limitation of round robin algorithm. It is based on the procedure that provides solution to eliminate the flaws of round robin due to the usage of static time quantum. In it the operating system finds out the optimal value of time quantum according to the burst time of each waiting process that is found in the ready queue. The performance rate of AN algorithm is good as compare to existing round robin algorithm but not very good as compare to NK algorithm [20].

NK Algorithm

Kundargi and Bandekar [20] proposed an algorithm named as NK Algorithm in order to improvise existing round robin algorithm. In it the time quantum is computed dynamically by choosing the burst time depending on the set of available processes. The idea of this algorithm is that in the first step time slice value is equivalent to burst time of the arrived process. If at the same time many processes arrived then time slice is initialized by computing the average burst time. In this condition each process is organized in the ready queue and the process scheduling proceeds on the basis of First-Come First - Serve. The NK algorithm is better than AN [19] algorithm due to better time quantum adjustment. This algorithm provides better results of turnaround time, response time and minimum waiting time as compared to RR and AN algorithm.

IRR Scheduling

Nayak et al [21] proposed improvement to the performance of round robin by using dynamic time quantum. In this improvement median of burst time of the processes is calculated. Average number of context switches is better. The algorithm provided state of the art performance in case of average turnaround time and average waiting time.

CPU utilization by providing priority to processes with short burst time

Mythili et al [22] tried to improve CPU utilization by providing priority to processes with short burst time. Process is allowed to execute and does not have to wait for next burst time if the burst time is less than 2X of the time quantum but more than 1X of the time quantum.

CMRR

Reddy et al [23] proposed an improvement to existing round robin by calculating the mean of the given processes namely check mean with round robin (CMRR). The proposed technique decreases the context switch which reduces the average waiting time and average turnaround time as compared to the existing IRR technique [21].

PROPOSED CPU SCHEDULING ALGORITHEM

The proposed randomized dynamic quantum (RDQ) scheduling algorithm is based on the idea of arranging incoming processes in ascending order according to burst time in the ready queue. The RDQ algorithm utilizes dynamic time quantum to allocate all processes to the CPU one by one for the completion of their execution. The core algorithm is well practiced algorithm, the round robin algorithm. Simulation results show that proposed algorithm has better performance rate as compared to other CPU scheduling algorithms. The RDQ scheme is explained as Algorithm 5 that consists of the following steps:

1. Take out the set of processes and their burst time.
2. By using any sorting technique arrange the processes in increasing order with respect to their relative burst time and assign process numbers after sorting.
3. Assign minimum=middle process burst time and maximum=last process burst time.
4. Generate time quantum randomly (between minimum to maximum).
5. Execute processes from first to last one by one according to time quantum.

Repeat step 4 and 5 until all the processes successfully executed. Update process numbers after successful execution of each process.

ALGORITHM 5	**RDQ Scheduling**
1	Create an array **Rem_B_T[ ]** to keep track of remaining burst time of processes. Rem_B_T[*i*]=B_T[*i*];
2.	Create another array **W_T[ ]** to store waiting times of processes. Initialize this array as W_T[*i*]=0;
3.	Sort all processes in ascending Order according to B_T[*i*]; // assign process number after sorting
4.	Initialize **Min** = B_T[N/2];
5.	Initialize **Max** = B_T[N];
6.	Initialize time: **t = 0**;
7.	While all processes are not done {
	Qtm$_T$= Random # greater than **Min** and less than **Max**
	For each process *i* do following {
	If Rem_B_T[*i*] >Qtm$_T$
	t = t + Qtm$_T$
	Rem_B_T[*i*] = Rem_B_T[*i*] - Qtm$_T$;

	Else // Last cycle for this process
	$t = t + Rem\_B_T[i]$;
	$W_T[i] = t - B_T[i]$; $Rem\_B_T[i] = 0$; // This process is over
	Decrement **N;}}** // update remaining process numbers
8:	Compute turnaround time: $TA_T[i] = W_T[i] + B_T[i]$;

RDQ Working: An Example

In this section, we explain the proposed scheme using an example. Consider the set of processes P1-P5 with their relative burst time where arrival time is zero. Sorting is applied on them according to their burst time. The passes are designed to generate the time quantum randomly and then execute processes from first to last one by one according to their time quantum.

Process Name	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12

Sorting:

Process Name	P3	P4	P1	P5	P2
Burst Time	3	7	10	12	29

First Pass

1. Generate time quantum randomly (e.g. quantum=11)
2. Execute process from first to last one by one according to time quantum

Process Name	P3	P4	P1	P5	P2
Burst Time	3-3	7-7	10-10	12-11	29-11

Sorting:

Process Name	P5	P2
Burst Time	1	18

Second Pass

1. Generate time quantum randomly (e.g. quantum=14)
2. Execute process from first to last one by one according to time quantum

Process Name	P5	P2
Burst Time	1-1	18-14

Sorting:

Process Name	P2
Burst Time	4

Third Pass

1. Generate time quantum randomly (e.g. quantum=20)
2. Execute process from first to last one by one according to time quantum

Process Name	P2
Burst Time	4-4

Finally the last process is finished its execution. The Gantt chart representation of RDQ algorithm is given below:

P3	P4	P1	P5	P2	P5	P2	P2
3	7	10	11	11	1	14	4

EXPERIMENTAL ANALYSIS

To compare and contrast the performance of the proposed RDQ CPU scheduling algorithm, a scheduling algorithms simulator was developed and used to check the performance of the proposed methodology. In this section the results and details along with observations are presented.

Results

New proposed RDQ CPU scheduling algorithms is compared with FCFS, SJF, RR and NK CPU scheduling algorithms. For each comparison, different numbers of processes along with their burst time are taken and then they are scheduled. The numbers of processes in each list were 10, 50, 100, 150, 200, 300, 400 and 500.

Figure 1 and Figure 2 is the graphical representation of cumulative comparison of average waiting time and average turnaround time of proposed RDQ algorithm with FCFS, SJF, RR and NK scheduling algorithms. Using these analysis results, it can be easily concluded that RDQ algorithm shows marked improvement as compared to FCFS, RR and NK Algorithm.

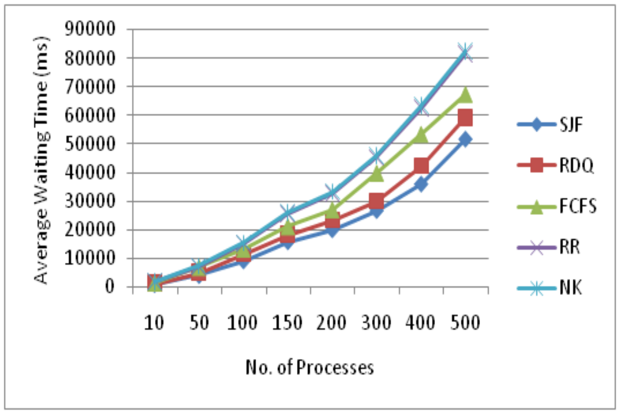

Figure 1. Cumulative comparison of average waiting time of scheduling algorithm

Performance analysis of RDQ

There are two major classes of performance criteria; the one is user-oriented criteria while the other is system oriented criteria. The parameters such as waiting time and turnaround time belong to user-oriented criteria and the CPU utilization belongs to system-oriented criteria.

Proposed scheduling algorithm has been implemented using C# language and executed several times using different process sets. The analysis of performance parameters are discussed here.

- Waiting Time: It was analysed that newly designed RDQ algorithm gives smaller value of average waiting time as compare to other existing scheduling algorithms.
- Turnaround Time: The proposed RDQ has minimum values of average turnaround time.
- CPU Utilization: RDQ also maximizes the processor utilization.
- Starvation: This problem is not present in RDQ scheduling algorithm for the reason that it provides equal time slice to every process for its execution.

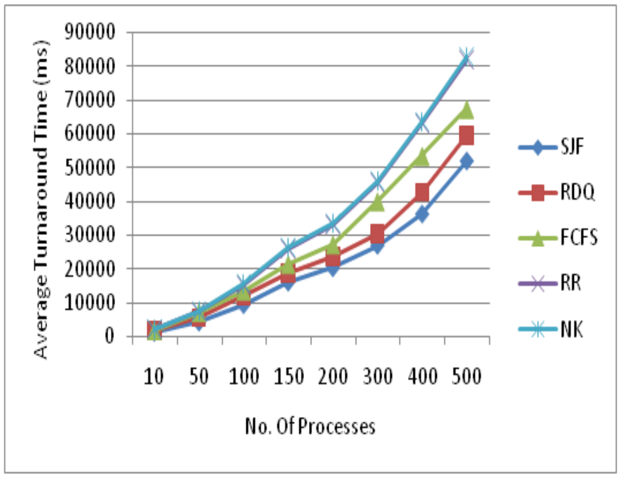

Figure 2. Cumulative comparison of average turnaround time of scheduling algorithm

- Be Fair: RDQ is suitable for time sharing systems as it provides fair share of processor to every process.
- Context Switching: RDQ does not suffer much due to overhead of context switching as compared to existing round robin scheduling algorithm for the reason that in the proposed scheduling algorithm, the dynamic quantum is utilized for the completion of processes execution.

The experimental results shown in the present section are used to derive performance metrics of each CPU scheduling algorithm presented in literature. The performance detail of each algorithm is given in Table 2. (Appendix-I)

DISCUSSION

The generated results in the preceding section clearly represent the position and performance of the proposed RDQ algorithm. Results show that FCFS algorithm is simplest and easy to implement but having poor performance, minimum estimated throughput, its related average waiting time and average turnaround is also maximum. FCFS also suffers from convoy effect. The SJF algorithm is optimal because it gives minimum values of average turnaround and average waiting time but still have problem of starvation. The RR algorithm is pre-emptive and appropriate for time sharing system. It uses quantum value that is utilized to allocate the CPU to all the incoming processes one by one. The main overhead of RR algorithm arises due to static quantum size. It degenerates to FCFS ones the quantum size is too large, on the other hand if the quantum is too small than RR faces out a problem of context switching time. The NK algorithm is also a variant of RR algorithm with dynamic quantum but still not shows too much efficiency. The performance of newly proposed RDQ algorithm is close to SJF but when RDQ is compared with other algorithms it gives better performance by giving smallest values for average waiting and average turnaround time. The RDQ has removed the problems such as starvation, convoy effect, and context switching at great extent. Performance evaluation and analysis proves that RDQ is good enhancement of round robin algorithm and secures good position above other scheduling algorithms.

Summary and Future Discussion

The main goal of operating system with multiprogramming feature is to share the system resources fairly between several users and jobs. This multiprogramming system mainly focuses on utilizing CPU in such a way that the system performance is increased. Scheduling is an important function of operating system that makes possibility of efficient resource sharing. All the computer resources are scheduled before use. CPU is one of the important resources of the computer system so its scheduling is vital that help out in the understanding of complex procedures and methods used to govern the ways in which jobs are executed by the CPU. Various algorithms are used to implement scheduling and every algorithm has its own characteristics that differentiate it from others.

Algorithms are evaluated by using various ways such as deterministic modeling, queuing models, and simulations. The most common techniques are simulations and deterministic modeling because of their provision of accurate results of the relevant scheduling algorithms. The simulation involves programming a specific model of the computer system whereas deterministic modeling uses the selected algorithms and workloads of a system to generate a number that evaluates an efficiency of each algorithm for that specific workload.

In the present research work a novel CPU scheduling algorithm named randomized dynamic quantum is designed for time sharing systems and is based on the idea of using dynamic quantum that change in each pass of the algorithm. The RDQ algorithm sorts the incoming processes in ascending order according to their relative burst time. it utilizes the statically generated time quantum to execute processes one by one. The objective of designing the RDQ algorithm is to increase the performance of the system and to overcome the limitations of the existing algorithms. The proposed algorithm achieves the desired goal.

Currently proposed RDQ CPU scheduling algorithm has been designed to schedule process on uniprocessor systems but in future it can be enhanced in order to schedule processes in multiprocessor systems.

REFERENCES

1. Z. Khan, M. Alam, R. A. Haidri. Effective Load Balance Scheduling Schemes for Heterogeneous Distributed System. International Journal of Electrical and Computer Engineering. 2017, 7(5).
2. N. Srilatha, M. Sravani, Y. Divya. Optimal Round Robin CPU Scheduling Algorithm Using Manhattan Distance. International Journal of Electrical and Computer Engineering. 2017, 7(6).
3. G. T. Hicham, E. A. Chaker, E. Lotfi. Comparative Study of Neural Networks Algorithms for Cloud Computing CPU Scheduling. International Journal of Electrical and Computer Engineering. 2017, 7(6).
4. A. Kumar, B. Alam. Energy Harvesting Earliest Deadline First Scheduling Algorithm For Increasing Lifetime of Real Time Systems. International Journal of Electrical and Computer Engineering. 2019, 9(1).
5. L. Dhanesh, P. Murugesan. A Novel Approach in Scheduling of the Real-Time Tasks in Heterogeneous Multicore Processor with Fuzzy Logic Technique for Micro-grid Power Management. International Journal of Power Electronics and Drive Systems, 2018, 9(1).
6. U. Saleem, M.Y. Javed. Simulation of CPU Scheduling Algorithms. TENCON, 2000, pp. 562-567.
7. A. Silberschatz, P.B. Galvin and G. Gagne, Operating System Concepts, 8th edition, 2009.
8. M. Milenkovic. Operating Systems Concepts and Design. McGraw Hill, IBM Corporation, 1992.
9. A.S. Tanenbaum, and A.S. Woodhull. Operating system design and implementation. 2nd Edition, 1997.
10. M.G. Nutt. Operating Systems. Addison Wesley, 2000.
11. R.J. Matarneh. Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Brust time of the Now Running processes. American Journal of Applied Sciences. 2009, 6(10), pp. 18311-1837.
12. A. Bashir, M.N. Doja and R. Biswas. Finding Time Quantum of Round Robin Cpu Scheduling Algorithm Using Fuzzy Logic. IEEE Internal Congference on Computer and Electrical Engineering, 2008.
13. T. Helmy, and A. Dekdouk. Burst Round Robin as a Proportional-Share Scheduling Algorithm. IEEE-GCC Conference on towards Techno-industrial Innovations, Bahrain, 2007, pp. 424-428.
14. R. Mohanty, H.S. Beheram, K. Patwari and M. Dash. Design and Performance Evaluation of a new Proposed Shortest Remaining Burst Round Robin (SRBRR) Scheduling Algorithm. International Symposium on Computer Engineering & Technology (ISCET), 17, 2010.
15. R. Mohanty, H.S. Beheram, K. Patwarim, M. Dash and M.L. Prasanna. Priority Based Dynamic Round Robin (PBDRR) Algorithm With Intelligent Time Slice for Real Time Systems. International journal of Advance Computer Science and applications, 2011, 2(2).
16. H.S. Behera. Weighted Mean Priority Based scheduling for interactive systems. Journal of global Research in computer science, 2011, 2(5).
17. H.S. Behera. Enhancing the CPU performance using a modified mean-deviation round robin scheduling algorithm for real time systems. Journal of global Research in computer science, 2012, 3(3).
18. L. Kishor and D. Goyal. Time Quantum Based Improved Scheduling Algorithms. International Journal of Advanced Research in Computer science and Software Engineering, 2013, 3(4).
19. A. Noon, A. Kalakech and S. Kadry. A new Round Robin based scheduling algorithm for operating system: Dynamic quantum using the mean average. International Journal of computer science, 2011, 8(3).
20. N. Kundargi and S. Bandekar. CPU Scheduling Algorithm Using Dynamic Time Quantum for Batch Systems.

International journal of latest trends in engineering and technology, 2013.
21. D. Nayak, S.K. Malla and D. Debadarshini. Improved Round Robin Scheduling using Dynamic Time Quantum. International journal of computer applications, 2012, 38(5).
22. N. Mythili,S. Pati, P. Korde and P. Dey. An advanced approach to traditional round robin CPU scheduling algorithm to prioritize processes with residual burst time nearest to the specified time quantum. Int. Conf. Ser.: Mater. Sci. Eng., 2017.
23. N. Reddy, S. Kumar and H. Santhi, P. Gayathri and N. Jaisankar. A New CPU Scheduling Algorithm Using Round-robin and Mean of the Processes, System and Architecture. ISBN: 978-981-10-8532-1, 2018, pp. 231-240.

Appendix-I

Table 2. Performance Metrics of Scheduling Algorithms

	FCFS	SJF	RR	NK	RDQ	IRR
Decision Mode	non pre-emptive	non pre-emptive	pre-emptive	Pre-emptive	pre-emptive	Pre-emptive
Turnaround Time	High	Less	less for short processes	less for short processes	slightly higher than SJF	Less than RR
Waiting Time	High	Less	less for short processes	less for short processes	slightly higher than SJF	Less than RR
CPU Utilization	Less	Less	High	High	High	High
Starvation	No	possible	No	No	No	No
Convoy Effect	Possible	No	No	No	No	No
Context switching	No	No	Yes	Yes	Yes Less than RR	Less than RR
Time Sharing System	No	No	Yes	Yes	Yes	Yes