IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

Design and Analysis of Fairness Aware Round Robin CPU Scheduling Strategy

Sumandeep Kaur Sidhu[1], Rakesh Kumar Bansal[2], Savina Bansal[3]

Research Scholar, Dept. of Electronics & Communication Engineering, GZSCCET, Bathinda, Punjab (India) [1]

Professor, Dept. of Electronics & Communication Engineering, GZSCCET, Bathinda, Punjab (India) [2, 3]

Abstract: Round Robin (RR), CPU scheduling algorithm, is widely accepted scheduling strategy for many time shared operating systems. The traditional Fixed quantum Round Robin (FRR) scheme works well for fair share scheduling though the large number of context switches lead to excessive system overhead. On the other hand, the available dynamic quantum based RR algorithm (DRR) reduces context switches though at the cost of fairness and results in service degradation to an individual process. In this paper, a new variant of RR named as adaptive Round Robin is proposed that trades off between fairness and context switches parameters in a judicious way and provide a better balancing among these conflicting parameters. The proposed algorithm chooses the time quantum adaptively based on existing burst time of the available jobs in the job pool. In this way, both smaller and larger size jobs get fair time for their execution. The performance of the suggested technique has been analyzed using extensive simulations on a wide variety of jobs. The paper also presents the comparative analysis of proposed algorithm with existing FRR and DRR scheduling algorithms on the basis of varying time quantum, average waiting time, average turnaround time, performance ratio and number of context switches.

Keywords: CPU scheduling, Round Robin CPU scheduling algorithm, Turnaround time, Waiting time, Context switching, Performance ratio, Simulation analysis.

I. INTRODUCTION

Operating System allocate computing resources among the potentially competing requirements of multiple processes in an optimal way is known as scheduling. Scheduling is the most repetitively used fundamental concept in operating system.

It deals with allocation and arrangement of jobs to the processors, with the objective to reduce overall execution time of all the jobs. It is a key feature in multitasking, multiprocessing and real-time operating system design, in which, it is also necessary to allocate the resources in a fair manner to all the competing processes/jobs.

Scheduling is of two types: Non-preemptive and Preemptive scheduling. In non-preemptive scheduling, CPU is assigned to a process until it's execution is completed, while in the latter, running process is forced to release the CPU by the newly arrived process [SGG]. Efficient resource utilization is achieved by sharing system resources amongst multiple users and system processes.

Optimum resource sharing depends on the efficient scheduling of competing users and system processes for the processor, which renders process scheduling an important aspect of a multiprogramming operating system**.** Over the years, scheduling has been the focus of intensive research, and many algorithms have been developed.

CPU scheduling is a technique used by computer operating systems to manage the usage of the computer's central processing unit and it is the basis of multiprogramming systems. Almost all computer resources are scheduled before use.

The processes are scheduled according to the given burst time, arrival time and their priority. The number of resources including Memory, CPU, etc is used for execution of processes. Scheduler selects the ready processes from memory and allocates resource/CPU as per requirements.

When there are multiple ready processes, it is the CPU scheduling which decides which processes should be run. Whenever one process waits for some other resource, scheduler selects next process and allocates CPU to it. This process continues till the system request for termination of execution and then the last CPU burst ends up with it.

Allocating CPU to a process requires careful awareness to assure justice and avoid process starvation for CPU. Number of scheduling algorithms has been suggested with the aim to reduce: turnaround time, response time, average waiting time and the number of context switches. For job scheduling, commonly used algorithms are: First-Come First-Served (FCFS) Scheduling, Shortest-Job-First (SJF) Scheduling, Priority scheduling and Round Robin (RR) with varying degree of performances. However Round Robin scheduling algorithm is quite popular for timesharing and real time operating systems [**6**]. These

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

scheduling algorithms are widely used in communication networks and in operating systems to allocate resources to competing tasks. In the recent years, several researchers have tried to improve upon the time slice value which is the most crucial parameter in RR and proposed to choose time slice as dynamic and adaptive. Following section deals with some of the suggested expressions used by researchers with varying degree of performance.

These are Median method-based time quantum **[3]**, Smart Time Slice based method **[6]**, Median method based on threshold value of time quantum **[7]**, Dynamic Time Quantum based method **[9]**, and proposes a new adaptive round robin algorithm.

II. PROPOSED ALGORITHM

A. Adaptive Round Robin CPU Scheduling Algorithm
The proposed 'Adaptive Round Robin' (ARR) strategy based on fair-share-scheduling provides a tradeoff between the fixed quantum RR algorithm and the Dynamic Round Robin.

It chooses the time quantum adaptively based on existing Burst Time (BT) of the available jobs in the job pool. Initially, on the basis of minimum and maximum job burst time, a small fixed time quantum is selected so that all available jobs get a fair share of CPU time without too much waiting.

In the next round, to provide higher time quantum to CPU intensive jobs, time quanta is incremented in an exponential way. This process is repeated in every round until and unless the ready queue becomes empty. In this way, both smaller and larger size jobs get a fair time slice for their execution.

Time Quantum (at round n)= (Min BT +Max BT)/2+(Min BT +Max BT)/(8-n-1)
Where n is no. of rounds (1, 2, 3 ……)

The performance has been analyzed using extensive simulations on a wide variety of jobs where performance analysis has been carried out in terms of Turnaround time, Waiting time, context switches and performance ratio.

Turnaround Time is total time (from entry to exit) which is spent to complete the process. Waiting Time is the total time that a process has been waiting in ready queue before the allocation of processor/CPU.

Context Switch is process of storing and restoring context (state) of a preempted process, so that execution can be resumed from same point at a later time. Performance Ratio (PR) is the ratio of turnaround time to service time. This value indicates the relative delay experienced by a process. The minimum possible value for this ratio is 1.0; increasing values correspond to a decreasing level of service fairness. PR= Tr/Ts, where Tr is Total turnaround time and Ts gives Total service time

B. Case Study
The performance of the proposed algorithm ARR has been compared with the available traditional Fixed RR and Dynamic RR algorithms **[10]** which choose TQ as follows:

For the FRR algorithm, the TQ remains fixed in all rounds (let us take it as 20% of the highest burst time in the initial job queue).

Whereas, for the DRR algorithm considered **[10],** it keeps varying in every round.
Step 1: Arrange the jobs in ascending order in a queue Q,
Step2: let Y= process name in the Q, and n= number of processes
Step3: let M denote median = Y(n+2)/2if n is odd and M=1/2 (Yn/2 + Yn/2+1), if n is even)**.**
Step4: TQ_{DRR}= [M + highest burst time]/2,

The ARR shows better results in ascending, descending and random order and in this research paper, random order (worst case) is taken for ARR.

C. Example
We consider five processes P1, P2, P3, P4 and P5 arriving at time 0 with burst time 26, 82, 70, 31 and 40 respectively shown in Table 1. Table 2 shows the experimental result of RR, DRR and the proposed algorithm ARR.

Table 1: Data in Random Order

Process	Arrival Time	Burst Time
P1	0	26
P2	0	82
P3	0	70
P4	0	31
P5	0	40

Fig1 illustrate the working of these three algorithms individually with the help of Gantt chart for the data available in Table 1.

Table 2: Results for the case example using ARR, DRR and FRR alg

Jobs	Algorithm	TQ	Avg WT	Avg TAT	Avg PR	CS
5	ARR	**40.5,45**	**112.3**	**162.1**	**3.23**	**6**
	DRR	61,15,6	133.4	183.2	3.75	7
	FRR	20	137.4	187.2	4.03	14

DOI 10.17148/IJARCCE.2016.51160

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

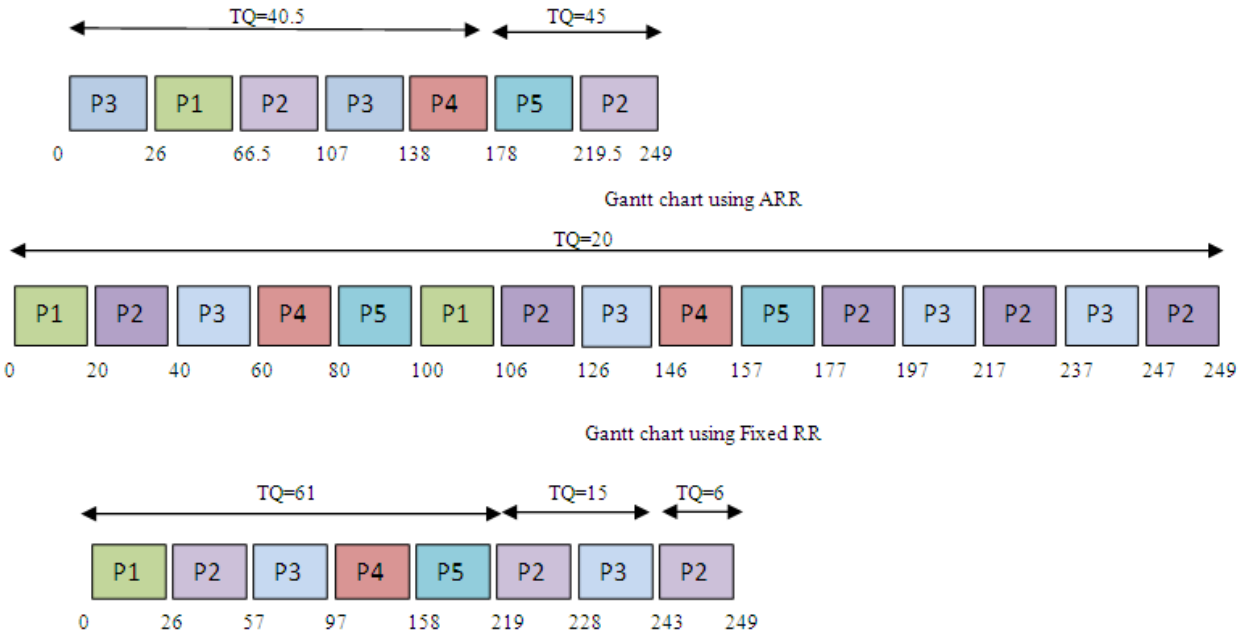
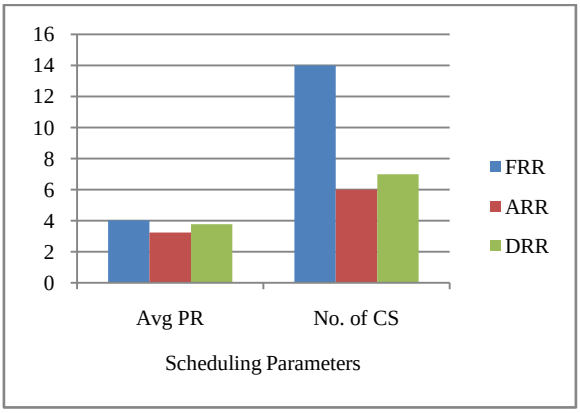


Fig1. Gantt chart using ARR, FRR and DRR (for Table 1)



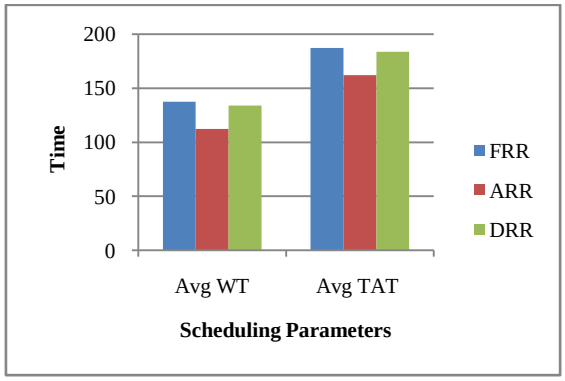


Fig 2: Performance Comparison of FRR, ARR and DRR

From Fig 2, it has been observed that the proposed ARR scheduling algorithm gives better results than Fixed RR and DRR scheduling algorithms in terms of reducing fairness degradation (PR) and context switches. Further the proposed ARR algorithm also reduces average waiting time and turnaround time to a great extent. The main motive for developing this algorithm to improve the performance of the system in terms of reducing fairness degradation and number of context switches fulfils whereas Fixed RR and DRR fail to do so.

Table 3 shows the individual performance PR of each process using RR, DRR and ARR. Further the results of Table 3 are shown in line graph in Fig 3.

Table 3: Individual performance ratio of each process

Jobs	Burst Time	ARR	DRR	RR
P1	26	1	1	4.077
P2	82	2.677	3.037	3.037
P3	70	3.557	3.514	3.529
P4	31	4.452	5.774	5.065
P5	40	4.45	5.475	4.425

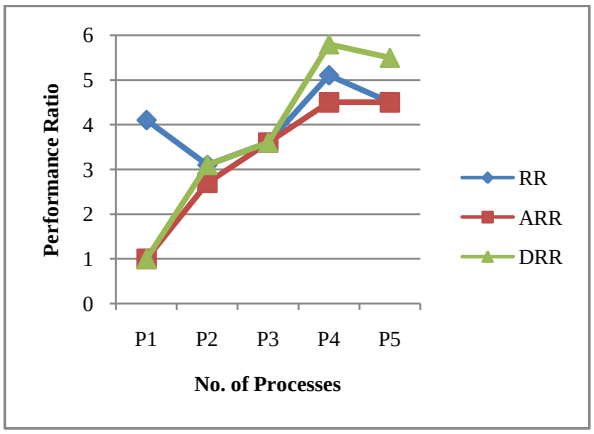


Fig 3: Individual performance ratio of each process

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

III. SIMULATION BASED PERFORMANCE ANALYSIS

The proposed algorithm is analyzed using large number of job sets to ascertain its effectiveness and the simulation parameters used for the data set is as shown in Table 4. This section focuses on the overall results i.e. extract from extensive simulations on a wide variety of jobs. In each case, the proposed algorithm's results can be compared with the results of traditional FRR and DRR algorithms.

Table 4: Simulation parameters

No. of Jobs	50, 100, 200, 500
Execution time	25-100; 25-500; 25-800; 25-1000; 25-1500; 25-2500; 25-10000; 25-25000; 25-50000; 25-100000; 25-2500000; 25-5000000; 25-10000000
Arrival Time	0

A. No. of processes=50

Table 5: Comparison of fairness degradation and no. of CS using ARR and DRR alg. w.r.t. FRR alg

Sr.No.	Jobs_minET-maxET	ARR (% degradation)		DRR (%degradation)	
		PR	CS	PR	CS
1	50_25-100	16	50	28	64
2	50_25-500	6	41	9	58
3	50_25-800	2	40	3	58
4	50_25-1000	0	39	0	57
5	50_25-1500	2	40	-1	57
6	50_25-2500	2	40	-1	57
7	50_25-10000	-1	39	-5	57
8	50_25-25000	0	39	-5	57
9	50_25-50000	-1	39	-5	57
10	50_25-100000	-1	39	-5	57
11	50_25-2500000	-1	39	-5	57
12	50_25-5000000	-1	39	-5	57
13	50_25-10000000	-1	39	-5	57
	Avg degradation	**1.7**	**40.24**	**0.15**	**57.7**

From Table 5, the calculations shows that the average degradation of proposed ARR algorithm is 1.7% better while DRR algorithm is only 0.15% better than FRR algorithm and improvement in average context switches of ARR is 40.24% better while DRR is 57.7% better than the than FRR algorithm. The above results are plotted in bar graph form are shown in Fig 4.

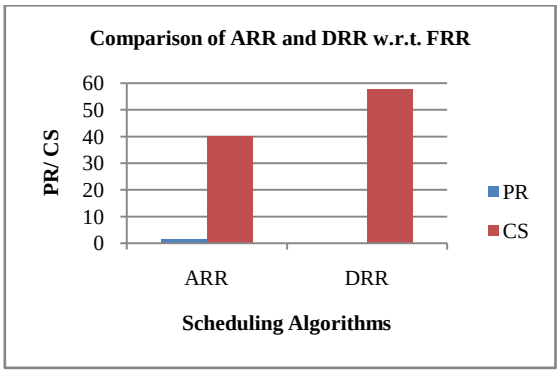


Fig 4: Comparison results using ARR and DRR w.r.t FRR

B. No. of processes= 100

Table 6: Comparison of fairness degradation and no. of CS using ARR and DRR alg. w.r.t. FRR alg

Sr.No.	Jobs_minET-maxET	ARR(% degradation)		DRR(%degradation)	
		PR	CS	PR	CS
1	100_25-100	14	51	23	63
2	100_25-500	-3	41	-11	57
3	100_25-800	-9	39	-22	56
4	100_25-1000	-12	39	-27	56
5	100_25-1500	-15	39	-34	56
6	100_25-2500	-19	38	-43	56
7	100_25-10000	-33	38	-73	55
8	100_25-25000	-39	38	-85	55
9	100_25-50000	-41	38	-91	55
10	100_25-100000	-43	38	-94	55
11	100_25-2500000	-44	38	-98	55
12	100_25-5000000	-44	38	-98	55
13	100_25-10000000	-44	38	-98	55
	Avg degradation	**-25.54**	**39.62**	**-57.77**	**56.08**

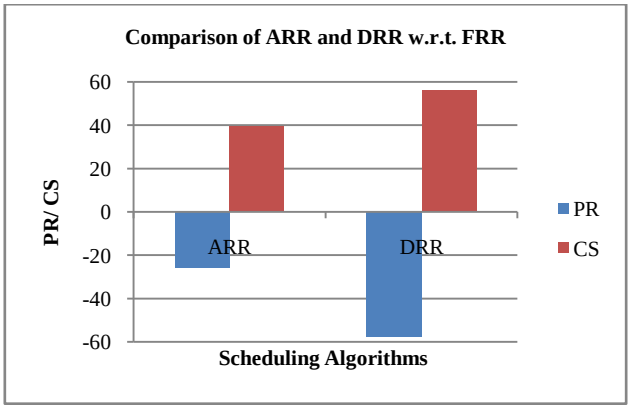


Fig 5: Comparison results using ARR and DRR w.r.t. to FRR

In case of 100 processes, Table 6 and Fig 5 shows that the average fairness degradation suffered by proposed ARR algorithm is only -25.54% while for DRR algorithm it is -

DOI 10.17148/IJARCCE.2016.51160

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 11, November 2016

57.77% with respect to FRR algorithm and improvement in average context switches of ARR is 39.62% better while DRR is 56.08% better than FRR algorithm.
Similarly, the simulation results were obtanied for 200 and 500 job sets.

C. Overall degradation analysis

Table 7: Overall Avg. performance degradation of ARR and DRR w.r.t. FRR alg

Sr. No.	Jobs	ARR alg (% degradation w.r.t. FRR)		DRR alg (%degradation w.r.t. FRR)	
		PR	CS	PR	CS
1	50	1.7	40.24	0.15	57.7
2	100	-25.54	39.62	-57.77	56.08
3	200	-15.39	41.16	-35.31	59.47
3	500	-15.93	40.24	-38.39	56.7
	Overall Avg degradation	**-13.39**	**40.32**	**-33**	**56.47**

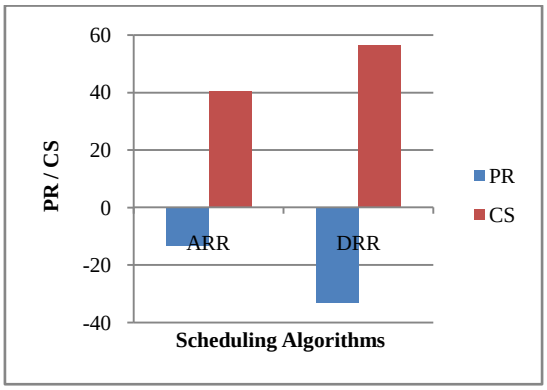


Fig 6: Overall average degradation w.r.t. FRR of ARR and DRR alg.

In Table7 and Fig 6, the negative value indicates more degradation in terms of performance ratio (unfairness of the jobs) and positive value shows betterness in terms of performance ratio and improvements in terms of context switches w.r.t to FRR and DRR.

❖ With respect to FRR, the overall average percentage degradation suffered by ARR algorithm for 50, 100, 200 and 500 jobs set is only -13.4% while for DRR algorithm it is -33%. This exhibits that ARR suffers 20% less degradation in terms of unfairness w.r.t. DRR algorithm.
❖ Similarly with respect to FRR, the overall percentage improvement in average context switches by ARR algorithm for 50, 100, 200 and 500 jobs set is 40.3%, while for DRR algorithm it is 57.5%.

IV. CONCLUSION

From the performance analysis, it can be gathered that the proposed ARR scheduling algorithm is better able to handle the fairness and context switches issue simultaneously in comparison to the traditional and available FRR and DRR scheduling algorithms.

FUTURE SCOPE

Time Quantum is the bottleneck facing RR algorithms. In this paper, the main concentration in proposing ARR scheduling algorithm is to choose the time quantum adaptively in order to reduce fairness ratio and number of context switches which are the most crucial parameters of upcoming works. Recent works are giving more importance to fair scheduling practices and also the individual service degradation suffered by various jobs. So, there is a need to develop more improvements in RR scheduling algorithm to solve the problem of time quantum chosen in order to reduce fairness ratio and context switches overhead.

REFERENCES

[1] Mohammed Abdullah, Hassan Al Hagery, “A selective Quantum of Time for RR Algorithm to increase CPU Utilization”, International Journal of Computer Information Systems, Vol. 3, No. 2, Qassim, Kingdom of Saudi Arabia, August, 2011.
[2] Ajit, S, Priyanka, G and Sahil, B, “An Optimized Round Robin Scheduling Algorithm for CPU Scheduling”, International Journal on Computer Science and Engineering (IJCSE), Vol. 02, No. 07, pp 2382-2385.
[3] H.S.Behera, R.Mohanty, Debashree Nayak. “A New Proposed Dynamic Quantum With Re-Adjusted Round Robin Scheduling Algorithm and Its Performance Analysis “International Journal of Computer Applications (0975 – 8887) Vol. 5, No.5, August 2010.
[4] H. M. Chaskar and U. Madhow, “Fair scheduling with tunable latency: a round robin approach”, IEEE/ACM Trans. Net., 11(4):592–601, 2003.
[5] Tarek Helmy, Abdelkader Dekdouk, “Burst Round Robin as a Proportional-Share Scheduling Algorithm”, IEEE Proceedings of the fourth IEEE-GCC Conference on towards Techno-Industrial Innovations, pp. 424-428, Bahrain, November 2007.
[6] Saroj Hiranwal and K. C. Roy “Adaptive Round Robin Scheduling Using Shortest Burst Approach Based On Smart Time Slice” International Journal of Computer Science and Communication Vol. 2, No. 2, pp. 319-323 July-December 2011.
[7] Rami J. Matarneh, "Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes", American Journal of Applied Sciences Vol.6, No. 10, pp. 1831-1837, 2009.
[8] Manish Kumar Mishra and Abdul Kadir Khan, “An Improved Round Robin CPU Scheduling Algorithm” JGRCS Journal of Global Research in Computer Science ISSN: 2229-371X, Volume 3, No. 6, June 2012.
[9] Abbas Noon, Ali Kalakech Seifedine Kadry, “A New Round Robin Based Scheduling Algorithm for Operating System: Dynamic Quantum Using the Mean Average” IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May 2011.
[10] Debashree Nayak, Sanjeev Kumar Malla, Debashree Debadarshini, “Improved Round Robin Scheduling using Dynamic Time Quantum” International Journal of Computer Applications (0975-8887), Volume 38-No.5, January 2012.
[11] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, “Operating System Concepts”, 8th edition.`