Improved Variable Round Robin Scheduling Algorithm

Ayush Mangukia - 191IT211
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email:ayushmangukia.191it211@nitk.edu.in

Mohammed Ibrahim - 191IT230
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email:mibrahim.191it230@nitk.edu.in

Soorya Golamudi - 191IT252
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email: sooryahyma.191it252@nitk.edu.in

Nishant Kumar - 171EE230
Electrical and Electronics
National Institute of Technology Karnataka
Surathkal, India 575025
Email: nishantkumar.171ee230@nitk.edu.in

Anand Kumar M
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
Email: m_anandkumar@nitk.edu.in

***Abstract*—Process scheduling allocates different processes (or threads) to the CPU. The type of scheduling algorithm affects the efficiency of the system. It allocates processes in a specific order to optimize those functions. There are various types of scheduling algorithms, one of them is the Round Robin algorithm.**

In Round Robin each process is given a certain Time Quantum (TQ), which means each process will have the same amount of time as the others without any bias. As there is no priority among the functions there is a relatively low response time in the CPU.

Value of the Time Quantum cannot be too small as the number of context switches will increase and lower the performance whereas if TQ is too huge it can prove negative to the ART.

The paper is about enhancing the Round Robin method to be a little more viable and efficient.

Keywords — CPU scheduling, process scheduling, round robin scheduling, variable quantum, scheduling algorithms, average waiting time, average turnaround time, average response time, context switches.

I. Introduction

CPU or Process Scheduling assigns processes to the CPU in determined order to enhance some objective functions. It is considered one of the most crucial parts in any operating system. The scheduling algorithm used affects the performance of the system.

Several CPU scheduling algorithms have been implemented to meet a varied number of OS requirements, such as Round Robin (RR), Shortest Job First (SJF), Priority, First Come First Served (FCFS), and Shortest Remaining Time First (SRTF). In this work, we focus on Round Robin (RR) scheduling algorithm.

Round Robin is a commonly used process scheduling algorithm and is preferred for its fairness towards all processes. It is a preemptive algorithm based on the principle of a time quantum (TQ). For each ready process waiting for execution, a certain time will be assigned equally and in circular fashion, treating all processes fairly without priority. It is highly preferred in time-sharing and real-time OS's. As each process is being given a fixed amount of time there will be a relatively low response time.

However, in spite of its advantages, RR algorithm produces high average waiting time, high average turnaround time, and a large number of context switches. TQ plays a critical role in the behavior of the RR approach. Using a small value of TQ produces a lot of context switches which reduces the system performance, while, using larger value of TQ affects the average response time negatively.

In this paper, we look upon the analysis of a new improved approach named, the Eighty-Five Percentile Round-Robin algorithm (EFPRR), to further improve the performance of RR based algorithms. Extensive experiments have been made to examine the performance of the proposed approach. Finally, this paper presents comprehensive comparisons between the proposed approach for different types of data-sets and variable characteristics. To facilitate this work, we run the code through various values for each variable and changing the variables itself.

We also look upon the Aspect of Parallel Processing by efficiently giving tasks to different processors to increase productivity and decrease waiting times.

II. Literature Survey

This section provides a short insight of several different process scheduling algorithms available right now.

The classical methods include First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling and Round Robin Scheduling. The problem with FCFS is waiting times are too long if a big process comes in the Queue. Similarly, SJF causes starvation if short processes keep coming. Even Priority causes starvation when important processes keep coming and lower priority processes may be postponed indefinitely.

Round Robin Scheduling Algorithm has the drawback of causing high waiting time for each process if the quantum is set too small. The quantum is set forehand and is generally not changed. There are many papers which have some improvements on the Round Robin Algorithm. One such improvement accredited by many authors was to check if the remaining time of the process under execution is less than the Time Quantum, then the processor is re-assigned to the process for the remaining time. There was another improvement which spoke about setting the time value of Time Quantum according to the Data Set. It used the approach of setting the Quantum value as the Twice the Average of Burst Times i.e. Two Times the Mean or set the Quantum Value as the middle most burst time in the Data Set i.e. Median.

III. Data - Set Used

5 Synthetic Data Sets of different sizes for fair comparisons to assess the performance of the algorithms have been used. The data sets have a variation of different Burst and Arrival Times.

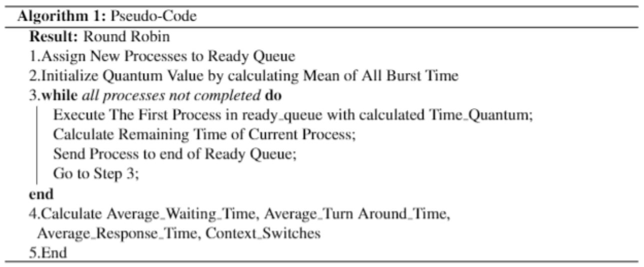
TABLE I
CHARACTERISTICS OF DATASETS

	Dataset	**Number of Processes**	**Min and Max Burst Time (seconds)**	**Min and Max Arrival Time (seconds)**
1	dataset - 50	50	1, 12	0, 10
2	dataset - 70	70	1, 10	0, 15
3	dataset - 100	100	1, 13	0, 30
4	dataset - 150	150	1, 9	0, 60
5	dataset - 200	200	1, 10	0, 75

IV. Algorithm Used

The proposed algorithm further improves the RR algorithm by setting a variable quantum and implementing Parallel Processing to efficiently divide the processes to different processors.

The classical RR Algorithm has a fixed quantum value for the whole time and doesn't change it according to the ready queue. In the proposed algorithm variable time quantum which is calculated using the processes only in the ready queue is implemented. All the processes are sorted in increasing order of burst time so that the shortest job comes first in the ready queue.

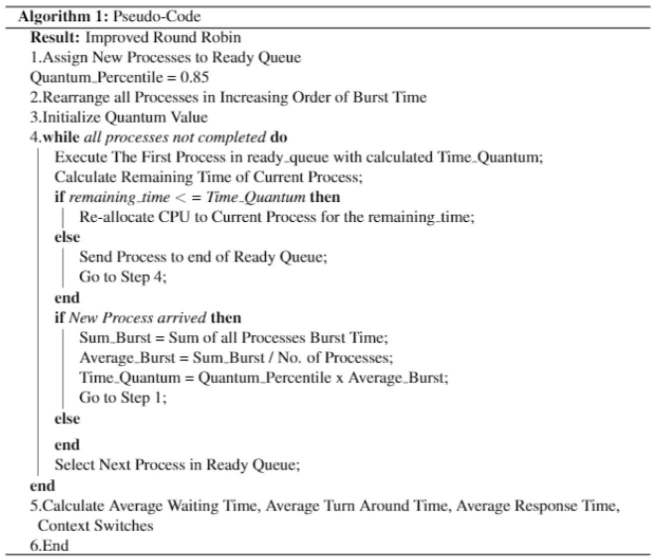


```
Algorithm 1: Pseudo-Code
  Result: Round Robin
  1.Assign New Processes to Ready Queue
  2.Initialize Quantum Value by calculating Mean of All Burst Time
  3.while all processes not completed do
      Execute The First Process in ready_queue with calculated Time_Quantum;
      Calculate Remaining Time of Current Process;
      Send Process to end of Ready Queue;
      Go to Step 3;
  end
  4.Calculate Average_Waiting_Time, Average_Turn Around_Time,
    Average_Response_Time, Context_Switches
  5.End
```

Fig. 1. Round Robin Algorithm

If the remaining time for that process is less than the quantum value then the same process is run again. Else, it is sent to the end of the ready queue.

If a new process arrives then the Quantum value is re-calculated and repeat the same process again.

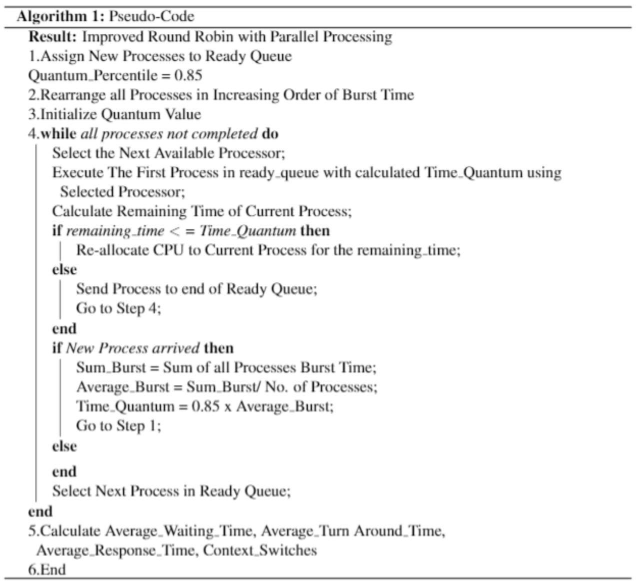


```
Algorithm 1: Pseudo-Code
  Result: Improved Round Robin
  1.Assign New Processes to Ready Queue
  Quantum_Percentile = 0.85
  2.Rearrange all Processes in Increasing Order of Burst Time
  3.Initialize Quantum Value
  4.while all processes not completed do
      Execute The First Process in ready_queue with calculated Time_Quantum;
      Calculate Remaining Time of Current Process;
      if remaining_time < = Time_Quantum then
          Re-allocate CPU to Current Process for the remaining_time;
      else
          Send Process to end of Ready Queue;
          Go to Step 4;
      end
      if New Process arrived then
          Sum_Burst = Sum of all Processes Burst Time;
          Average_Burst = Sum_Burst / No. of Processes;
          Time_Quantum = Quantum_Percentile x Average_Burst;
          Go to Step 1;
      else
      end
      Select Next Process in Ready Queue;
  end
  5.Calculate Average Waiting Time, Average Turn Around Time, Average Response Time,
    Context Switches
  6.End
```

Fig. 2. Improved Round Robin Algorithm

The same algorithm can be implemented with Multiple Processors. It will use several systems or processors to run different processes in parallel which will significantly lower the values of time metrics and context switching even with 2 or 4 processors. This leads to better management of workload for the individual processors compared to regular parallel processing. This also results in better turn around times for the processes also compared to regular parallel processing.

Once the processor and process to be run is selected the algorithm runs in a similar fashion to the Improved Round Robin Algorithm. Process selection is also same as the algorithm used before where in the processes are sorted in the ready queue and a quantum is calculated accordingly. The selection of processor is done such that next processor to be free is assigned the process. If the process that a processor is running has remaining burst time less than the time quantum then it will be the free before the passage of a time quantum and accordingly the first processor the be free in the within the next time quantum is selected.



```
Algorithm 1: Pseudo-Code
 Result: Improved Round Robin with Parallel Processing
 1.Assign New Processes to Ready Queue
 Quantum_Percentile = 0.85
 2.Rearrange all Processes in Increasing Order of Burst Time
 3.Initialize Quantum Value
 4.while all processes not completed do
     Select the Next Available Processor;
     Execute The First Process in ready_queue with calculated Time_Quantum using
      Selected Processor;
     Calculate Remaining Time of Current Process;
     if remaining_time < = Time_Quantum then
         Re-allocate CPU to Current Process for the remaining_time;
     else
         Send Process to end of Ready Queue;
         Go to Step 4;
     end
     if New Process arrived then
         Sum_Burst = Sum of all Processes Burst Time;
         Average_Burst = Sum_Burst/ No. of Processes;
         Time_Quantum = 0.85 x Average_Burst;
         Go to Step 1;
     else
     end
     Select Next Process in Ready Queue;
 end
 5.Calculate Average_Waiting_Time, Average_Turn Around_Time,
  Average_Response_Time, Context_Switches
 6.End
```

Fig. 3. Improved Round Robin Algorithm with Parallel Processing

V. Algorithm Analysis

For easier understanding and visualization, we will look at the Gantt charts of scheduling processes from a minimal data set (with ten processes to be scheduled) for the following algorithms. In the case of improved round-robin for multi-processors, we will look at the implementation results on two cores and four cores.

TABLE II
Data Set for Gaant Chart

Process	Arrival Time	Burst Time
1	0	5
2	0	4
3	1	2
4	1	20
5	3	11
6	4	1
7	4	5
8	7	4
9	7	4
10	9	10

A. Round Robin Algorithm

In this algorithm, the time quantum ’q’ is calculated once in the beginning as 85% of the mean burst time of all the (10) processes. As seen in the Gantt chart, the processes are pre-empted irrespective of how much burst time is remaining. This is causing a significant rise in the number of context switches.

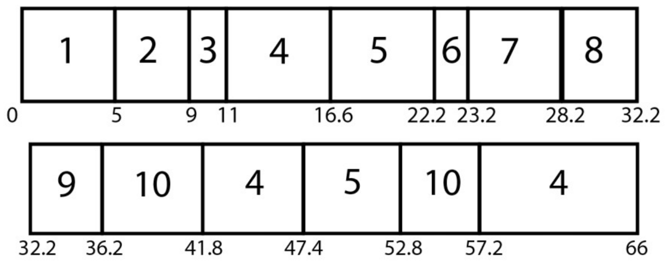


Fig. 4. Regular Round Robin (Gantt Chart)

B. Improved Round Robin Algorithm

In the improved version of the algorithm, the time quantum ’q’ is dynamically calculated as 85% of the mean remaining burst time of the processes currently in the ready queue. As depicted by the Gantt chart, the time quantum is optimized for currently waiting/executing processes. The algorithm also avoids unnecessary pre-emptions (when the remaining burst time is lesser than the current time quantum). This is significantly reducing the number of context switches.

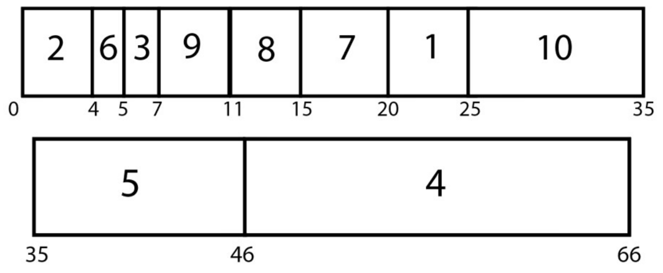


Fig. 5. Improved Round Robin (Gantt Chart)

C. Improved Round Robin Algorithm with Parallel Processing

This is the implementation of the same algorithm for more than one processor. The time quantum ’q’ is optimized for currently waiting/executing processes. As seen in the Gantt charts of the implementations for two cores and four cores, this allows more parallelism and significantly reduces the waiting time by executing the waiting processes in another processor instead of waiting for the current processor to be available.

1) Two Cores: As visualized by the Gantt chart, both processes are occupied simultaneously, which saves much more time than a single processor. There are also lesser number of context switches as the number of processes each processor has to execute is reduced.

2) Four Cores: Four cores allow even more parallelism and more distribution of processes amongst the processors. Consequently, the number of context switches is further reduced. This also significantly reduces the waiting time of the processes.

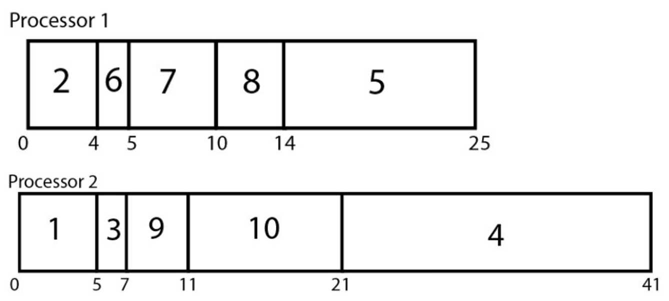


Fig. 6. Improved Round Robin - 2 Cores (Gantt Chart)

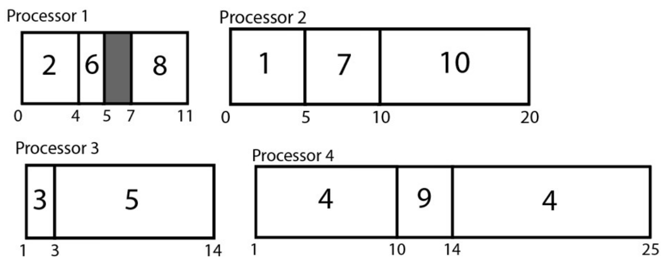


Fig. 7. Improved Round Robin - 4 Cores (Gantt Chart)

D. *Effect of number of Cores*

The increased number of processors available for parallelism significantly reduces the waiting time and context switches, thus allowing high performance in terms of execution time. However, there is an increased possibility of some processors being idle. This is also affected by the fact that many processes are sequential, and the algorithm does not allow the same process to be executed on different processors (simultaneously or otherwise). So specific scenarios can occur where one processor executes multiple processes with a significant burst time, and some other processor is idle.

VI. Performance Measures Used

There are several variables and output values present in the algorithms to compare their performances. The following 4 metrics have been selected:

A. *Average Waiting Time (AWT)*

Waiting time is the total time spent by the process in the ready state waiting for CPU. The Average Waiting Time (AWT) is calculated using Equation 1.

Waiting time = Turnaround time - Burst time

$$AverageWaitingTime = \frac{W_1 + W_2 + ... + W_N}{N} \tag{1}$$

$$W_1, W_2, \ldots, W_N$$

are the Waiting Times for all processes in the system.

B. *Average Turnaround Time (ATT)*

Turnaround Time refers to the entire time from the process' submission to its accomplishment. The Average Turnaround Time (ATT) is calculated using Equation 2.

Turnaround time = Completion time - Arrival time

$$AverageTurnaroundTime = \frac{TT_1 + TT_2 + ... + TT_N}{N} \tag{2}$$

$$TT_1, TT_2, \ldots, TT_N$$

are the Turnaround Times for all processes in the system.

C. *Average Response Time (ART)*

Response Time refers to the entire time spent between the first submission of the process to its initial implementation. The Average Response Time (ART) is calculated using Equation 3.

$$AverageResponseTime = \frac{RT_1 + RT_2 + ... + RT_N}{N} \tag{3}$$

$$RT_1, RT_2, \ldots, RT_N$$

are the Response Times for all processes in the system.

D. *Context Switches (#CS)*

Context Switches refers to the procedure of switching the CPU from one process to another. The efficiency increases as less Context Switches occur.

VII. Experimental Results

TABLE III
Performance Measures for different Quantum Values

Quantum Percentile	Number of Processes	AWT (seconds)	ART (seconds)	ATT (seconds)	#CS (switches)
0.1	50	165.216	17.461	171.276	439
0.2	50	144.522	31.109	150.581	198
0.3	50	126.380	47.674	132.440	124
0.4	50	117.690	58.606	123.750	97
0.5	50	111.779	66.769	117.839	79
0.6	50	101.7	81.360	107.76	64
0.7	50	97.410	87.87	103.47	59
0.8	50	95.408	90.668	101.468	56
0.85	50	93.814	92.374	99.874	53
0.9	50	93.120	93.120	99.180	49

It has been experimentally found that the following Quantum Percentile Values for different Data-Set size produces Low Average Waiting and Turnaround Time.

- For Data-Set of 50 processes, Quantum Percentile should be in Range 0.7 to 0.95
- For Data-Set of 100 processes, Quantum Percentile should be in Range 0.8 to 0.95
- For Data-Set of 150 processes, Quantum Percentile should be in Range 0.75 to 0.95
- For Data-Set of 200 processes, Quantum Percentile should be in Range 0.8 to 0.95

TABLE IV
PERFORMANCE MEASURES FOR EACH DATA TYPE USING DIFFERENT METHODS USING QUANTUM PERCENTILE AS 0.85

Method	Dataset Size	AWT (seconds)	ART (seconds)	ATT (seconds)	#CS (switches)
Round Robin	50	94.66864	176.18346	182.24346	82
Improved Round Robin	50	92.37462	93.81462	99.87462	53
Improved Round Robin Dual (2) Processor	50	43.24	43.24	49.3	48
Improved Round Robin Quad (4) Processor	50	21.2	21.14797	27.20797	49
Round Robin	70	117.518171	230.409029	235.851886	120
Improved Round Robin	70	121.2	121.2	126.642857	69
Improved Round Robin Dual (2) Processor	70	56.0142857	56.0142857	61.4571429	68
Improved Round Robin Quad (4) Processor	70	24.0857143	24.0857143	29.5285714	66
Round Robin	100	218.596035	424.161765	431.051765	173
Improved Round Robin	100	219.36	219.36	226.25	99
Improved Round Robin Dual (2) Processor	100	100.69	100.69	107.58	98
Improved Round Robin Quad (4) Processor	100	42.721415	42.66761	49.55761	97
Round Robin	150	235.049107	470.349307	475.475973	255
Improved Round Robin	150	239.78	239.78	244.906667	149
Improved Round Robin Dual (2) Processor	150	106.013333	106.013333	111.14	148
Improved Round Robin Quad (4) Processor	150	40.7887633	40.7679667	45.8946333	147
Round Robin	200	328.83582	639.1362	644.4512	332
Improved Round Robin	200	329.36	329.36	334.675	199
Improved Round Robin Dual (2) Processor	200	147.21	147.21	152.525	198
Improved Round Robin Quad (4) Processor	200	58.005	58.005	63.32	196

VIII. CONCLUSION

The new algorithms devised have been developed to include the prominent features of many suggestions for improvements to a round robin algorithm. The algorithms have provided positive output in regards to the metrics and can be of use in a real-time OS. The paper also has implemented an algorithm for mass processing of processes for larger projects by improving Parallel Processing.

IX. FUTURE WORK:

1) Using ML/DL concepts the prediction of the burst times of the processes can be implemented.
2) Splitting of large sequential processes to several smaller processes that can be run in parallel to further improve Parallel Processing.
3) Implementation in a real-life Operating System.
4) Modify the algorithm to allow more uniform distribution of processes amongst the processors to minimize their under-utilization.

X. ACKNOWLEDGMENT

The authors would like to acknowledge our guide, Dr. Anand Kumar M. for invaluable support, and we would also like to acknowledge numerous discussions on the subject with him.

REFERENCES

[1] OpenCV Documentation

[2] kaggle.com - for datasets

[3] H. Kopka and P. W. Daly, A Guide to LATEX

[4] Latex Table Generator
https://www.tablesgenerator.com/#

[5] Geeks For Geeks
https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/