WHATSAPP ANALYSER

A J-Component Project Report

Submitted by

Name: Dharmendra Kumar Sah (19BCE2633)
Chandan Thakur (19BCE2640)

Under the guidance of

Sudha S.

in partial fulfillment for the award of the degree of

B. Tech.

in

COMPUTER SCIENCE AND ENGINEERING

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

MAY 2020

CPU Scheduling with Dynamic Time Allocation and Predict Burst Time Using ML

Abstract

Scheduling is the central concept used in Operating Systems. It helps in choosing the processes for execution in an efficient sequence. Round Robin (RR) is one of the most commonly used CPU scheduling algorithms. But there is a degradation of performance with respect to context switching, which is an overhead cost to the system. The performance of a system depends on the choice of an optimal time quantum, so as to reduce context switching. In this paper, we have proposed a new variant of Round Robin which is better than the traditional Round Robin scheduling by reducing context switching, average waiting time and average turnaround time.

Index Term

CPU Scheduling, Round Robin, Response Time, Waiting Time, Turnaround Time

CPU Scheduling

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc. thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.

Round Robin

This is the preemptive version of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process

is completed during that time, then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Response Time

Response time is the time spent between the ready state and getting the CPU for the first time.

Waiting Time

Waiting time is the total time spent by the process in the ready state waiting for CPU.

Turnaround Time

Turnaround time (TAT) is the time interval from the time of submission of a process to the time of the completion of the process.

I. INTRODUCTION

CPU Scheduling is an essential operating system task, which basically allocates the CPU to a specific process for a given time slice. Scheduling requires precise instructions to ensure fairness and avoid process starvation. Operating systems can have up to 3 distinct types of schedulers such as a long-term scheduler, a mid-term and a short-term scheduler. The dispatcher is a module that gives control of the CPU to the process selected by the short-term scheduler. There are many CPU scheduling algorithms but the environment plays a crucial role in determining which scheduling technique is the most efficient. The most common measures to determine the efficiency of a scheduling algorithm are: Fairness, efficiency, response time, turnaround time, throughput.

II. LITERATURE SURVEY

Round Robin is the preemptive version of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time, then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution. So, we modified the traditional Round Robin algorithm reducing context switching, average waiting time and average turnaround time.

In Modified round robin algorithm, we can't say any it has any limitations or drawback. But of course, we can't deny the fact that context switching, average waiting time and average turnaround time can still be reduced. So, in coming future we are planning to reduce the context switching, average waiting time and average turnaround time even more. So that computer system will be more efficient.

III. PROPOSED METHODOLOGY

Introduction

In our proposed algorithm, we have to arrange the processes in ascending order according to their burst time present in the ready queue. Then the time quantum is calculated. For finding the optimal time quantum, median method is followed. The median can be found out by using the following formulae: Median = y = number located in the middle of a group of numbers arranged in ascending order n = number of processes. The standard deviation of the burst times is also a factor used in calculation of the dynamic time quantum.

$$\text{Median} = \begin{cases} Y_{\frac{n+1}{2}} & \text{if n is odd} \\ 1/2[Y_{n/2} + Y_{1+\frac{n}{2}}] & \text{if n is even} \end{cases}$$

y = number located in the middle of a group of numbers arranged in ascending order n = number

of processes. The standard deviation of the burst times is also a factor used in calculation of the dynamic time quantum.

To facilitate the same the following steps are followed of the algorithm designed:

1. Take the input of the processes and sort them according to their burst times.
2. Take the median and standard deviation of the burst times of the processes.
3. Calculate the dynamic quantum using the formula: Median – 0.5*(Standard Deviation of burst time)
4. Perform normal round robin scheduling on the given processes.
5. If a process is pre-empted at the end of the time quanta, send the process to a waiting queue.
6. After all processes are gone through, make the waiting queue into the executing queue and repeat steps 4,5 and 6 until all processes are completed.

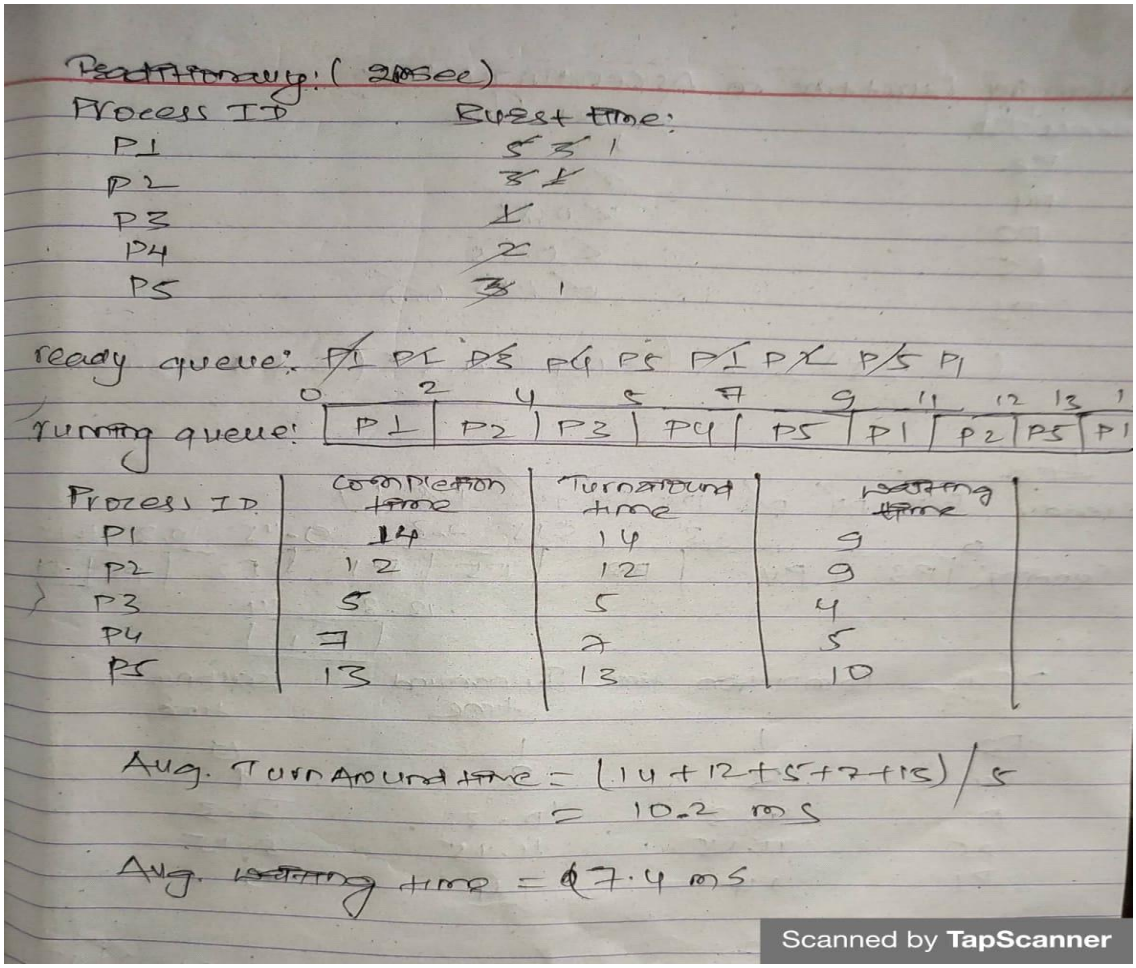
Framework. Architecture or Modules of the Proposed System:

Requirements: - Python 2.7/3.7

- Pandas
- Sklearn
- Matplotlib
- Numpy
- Seaborn
- Python IDLE/ Jupyter notebook
- Dev C/C++

IV. PROPOSED SYSTEM ANALYSIS AND DESIGN

Manually Solved Example

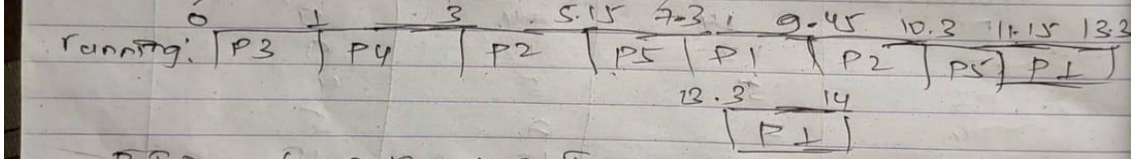


Arranging Burst time in Ascending order:

Process ID	Burst time
P3	1
P4	2
P2	3 0.85
P5	3 0.85
P1	5 2.85 0.1

(2.15) sec. (calculated - 0.5 * (S.No of burst time))

ready: P3 P4 P2 P5 P1 P2 P5 P1



P.ID	Completion time	Turnaround time	Waiting time
P3	1	1	0
P4	3	3	1
P2	10.3	10.3	7.3
P5	11.15	11.15	8.15
P1	14	14	9

Avg time = ?

Avg. turnaround time = (1+3+10.3+11.5+14)/5
= 7.89 ms

Avg. waiting time = (0+1+7.3+8.15+9)/5
= 5.09

V. IMPLEMENTATION

a) Implementation of Modified Round Robin

The first step, to run the Modified Round Robin using C

Requirements: - Dev C/C++

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct process {
    int process_id;
    int waiting_time,turnaround_time,burst_time,remaining_time;
}temp;
void ins(struct process *p,int n) {
    int i;
    for(i=0;i<n;i++) {
        printf("Enter the burst time of process P[%d]: ",i+1);
        scanf("%d",&p[i].burst_time);
        p[i].remaining_time=p[i].burst_time;
        p[i].process_id=i+1;
    }
}
int fcfs(struct process *p,int n) {
    int i;
    int total_time=0;
    printf("\nProcess Scheduling Order: ");
    for(i=0;i<n;i++) {
        p[i].waiting_time=total_time;
        total_time+=p[i].burst_time;
        p[i].turnaround_time=total_time;
        printf("P[%d], ",p[i].process_id);
    }
    printf("\n");
    display(p,n,total_time);
    return 1;
}
int sjf(struct process *p,int n) {
    int i,j;
    for(i=0;i<n-1;i++)
    for(j=0;j<n-1-i;j++)
    if(p[j].burst_time>p[j+1].burst_time){
        temp=p[j];
        p[j]=p[j+1];
        p[j+1]=temp;
    }
```

```
    int total_time=0;
    printf("\nProcess Scheduling Order: ");
    for(i=0;i<n;i++) {
        p[i].waiting_time=total_time;
        total_time+=p[i].burst_time;
        p[i].turnaround_time=total_time;
        printf("P[%d], ",p[i].process_id);
    }
    printf("\n");
    display(p,n,total_time);
    return 1;
}
int rr(struct process *p,int n) {
    int time_quanta=10;
    int total_time=0;
    int n1=n,count=0;
    printf("\nProcess Scheduling Order: ");
    while(n1) {
        if(p[count].remaining_time<=time_quanta && p[count].remaining_time>0){
            total_time+=p[count].remaining_time;
            p[count].remaining_time=0;
            p[count].turnaround_time=total_time;
            p[count].waiting_time=total_time-p[count].burst_time;
            printf("P[%d], ",p[count++].process_id);
            n1--;
        }
        else if(p[count].remaining_time>0) {
            p[count].remaining_time-=time_quanta;
            total_time+=time_quanta;
            printf("P[%d], ",p[count++].process_id);
        }
        else
        count++;
        if(count==n)
        count=count%n;
    }
    printf("\n");
    display(p,n,total_time);
    return 1;
}
int modifiedSch(struct process *p,int n) {
    int i,j,x,sum=0,t=n,n1=n,n2,total_time=0;
    struct process *ex_que=p,*wt_que;
    float mean,median,standard_deviation=0;
    for(i=0;i<n-1;i++)
    for(j=0;j<n-1-i;j++)
```

```
    if(p[j].burst_time>p[j+1].burst_time){
        temp=p[j];
        p[j]=p[j+1];
        p[j+1]=temp;
    }
    for(i=0;i<n;i++)
    sum+=p[i].burst_time;
    mean=(float)sum/n;
    if(n%2==0)
    median=(float)(p[n/2].burst_time);
    else
    median=(float)(p[(n-1)/2].burst_time+p[(n+1)/2].burst_time)/2;
    for(i=0;i<n;i++)
    standard_deviation+=(p[i].burst_time-mean)*(p[i].burst_time-mean);
    standard_deviation=sqrt(standard_deviation/n);
    int time_quanta=(int)(median-0.5*standard_deviation);
    printf("\nProcess Scheduling Order: ");
    while (t) {
        wt_que=(struct process*)malloc(sizeof(struct process)*n1);
        n2=0;
        for(i=0;i<n1;i++) {
            if(ex_que[i].remaining_time<time_quanta) {
                total_time+=ex_que[i].remaining_time;
                x=returnIndex(p,n,ex_que[i].process_id);
                p[x].turnaround_time=total_time;
                p[x].waiting_time=total_time-p[x].burst_time;
                ex_que[i].remaining_time=0;
                t--;
            }
            else{
                total_time+=time_quanta;
                ex_que[i].remaining_time-=time_quanta;
                wt_que[n2++]=ex_que[i];
            }
            printf("P[%d], ",ex_que[i].process_id);
        }
        ex_que=wt_que;
        n1=n2;
    }
    printf("\n");
    display(p,n,total_time);
    return 1;
}
int returnIndex(struct process *p,int n,int a) {
    int i;
    for(i=0;i<n;i++ ) {
```

```
        if(p[i].process_id==a)
        return i;
    }
    return -1;
}
int display(struct process *p,int n,int t) {
    float avg_wt=0,avg_tt=0;
    int i,j;
    for(i=0;i<n-1;i++)
    for(j=0;j<n-i-1;j++)
    if(p[j].process_id>p[j+1].process_id) {
        temp=p[j];
        p[j]=p[j+1];
        p[j+1]=temp;
    }
    printf("\nProcess_id\twaiting Time\tTurnaround Time\n");
    for(i=0;i<n;i++) {
        avg_wt+=p[i].waiting_time;
        avg_tt+=p[i].turnaround_time;
        printf("
P[%d]\t\t%d\t\t%d\n",p[i].process_id,p[i].waiting_time,p[i].turnaround_time);
    }
    avg_wt=(float)avg_wt/n;
    avg_tt=(float)avg_tt/n;
    printf("Average Waiting Time: %.2f\n",avg_wt);
    printf("Average Turnaround Time: %.2f\n",avg_tt);
    printf("Throughput: %f\n",(float)t/n);
}
int main() {
    int choice,n;
    while(1) {
        printf("\nCPU Scheduling Algorithms\n1. FCFS (First Come First Served)\n2. RR
(Round Robin)\n3. SJF (Shortest Job First)\n4. Modified Algorithm\n5. Exit\nEnter your
choice:");
        scanf("%d",&choice);
        if(choice==5) {
            printf("\nThank you\n");
            exit(0);
        }
        printf("Enter the number of processes: ");
        scanf("%d",&n);
        struct process p[n];
        switch(choice) {
            case 1: ins(p,n);
            fcfs(p,n);
            break;
```

```
            case 2: ins(p,n);
            rr(p,n);
            break;
            case 3: ins(p,n);
            sjf(p,n);
            break;
            case 4: ins(p,n);
            modifiedSch(p,n);
            break;
            default: printf("Wrong option selected");
        }
    }
    return 0;
}
```

b) Output

A. According to Round Robin

```
CPU Scheduling Algorithms
1. FCFS (First Come First Served)
2. RR (Round Robin)
3. SJF (Shortest Job First)
4. Modified Algorithm
5. Exit
Enter your choice:2
Enter the number of processes: 5
Enter the burst time of process P[1]: 12
Enter the burst time of process P[2]: 13
Enter the burst time of process P[3]: 14
Enter the burst time of process P[4]: 15
Enter the burst time of process P[5]: 16

Process Scheduling Order: P[1], P[2], P[3], P[4], P[5], P[1], P[2], P[3], P[4], P[5],

Process_id      waiting Time    Turnaround Time
 P[1]           40              52
 P[2]           42              55
 P[3]           45              59
 P[4]           49              64
 P[5]           54              70
Average Waiting Time: 46.00
Average Turnaround Time: 60.00
Throughput: 14.000000
```

B. According to Modified Round Robin

```
CPU Scheduling Algorithms
1. FCFS (First Come First Served)
2. RR (Round Robin)
3. SJF (Shortest Job First)
4. Modified Algorithm
5. Exit
Enter your choice:4
Enter the number of processes: 5
Enter the burst time of process P[1]: 12
Enter the burst time of process P[2]: 13
Enter the burst time of process P[3]: 14
Enter the burst time of process P[4]: 15
Enter the burst time of process P[5]: 16

Process Scheduling Order: P[1], P[2], P[3], P[4], P[5], P[2], P[3], P[4], P[5],

Process_id      waiting Time    Turnaround Time
 P[1]           0               12
 P[2]           51              64
 P[3]           51              65
 P[4]           52              67
 P[5]           54              70
Average Waiting Time: 41.60
Average Turnaround Time: 55.60
Throughput: 14.000000
```

Here, we can clearly notice the difference between average Waiting time and Turnaround time in these two algorithms.
In RR algorithm we find the Average waiting time is 46 unit while in Modified RR algo, it is just 41.6 unit.
Also, Average Turnaround time in RR algorithm is 60 unit whereas in Modified RR algorithm it is 55.6 sec.
Hence, we clearly notice that the modified algorithm is better than traditional RR algorithm.
Also, let us analyze some more outputs

```
CPU Scheduling Algorithms
1. FCFS (First Come First Served)
2. RR (Round Robin)
3. SJF (Shortest Job First)
4. Modified Algorithm
5. Exit
Enter your choice:2
Enter the number of processes: 4
Enter the burst time of process P[1]: 45
Enter the burst time of process P[2]: 14
Enter the burst time of process P[3]: 78
Enter the burst time of process P[4]: 26

Process Scheduling Order: P[1], P[2], P[3], P[4], P[1], P[2], P[3], P[4], P[1], P[3], P[4], P[1], P[3], P[1], P[3], P[3], P[3], P[3],

Process_id      waiting Time    Turnaround Time
 P[1]           80              125
 P[2]           40              54
 P[3]           85              163
 P[4]           74              100
Average Waiting Time: 69.75
Average Turnaround Time: 110.50
Throughput: 40.750000
```

```
CPU Scheduling Algorithms
1. FCFS (First Come First Served)
2. RR (Round Robin)
3. SJF (Shortest Job First)
4. Modified Algorithm
5. Exit
Enter your choice:4
Enter the number of processes: 4
Enter the burst time of process P[1]: 45
Enter the burst time of process P[2]: 14
Enter the burst time of process P[3]: 78
Enter the burst time of process P[4]: 26

Process Scheduling Order: P[2], P[4], P[1], P[3], P[1], P[3], P[3],

Process_id      waiting Time    Turnaround Time
 P[1]           72              117
 P[2]           0               14
 P[3]           85              163
 P[4]           14              40
Average Waiting Time: 42.75
Average Turnaround Time: 83.50
Throughput: 40.750000
```

c) **Now, let us see the output for other algorithms too:**

I. **FCFS**

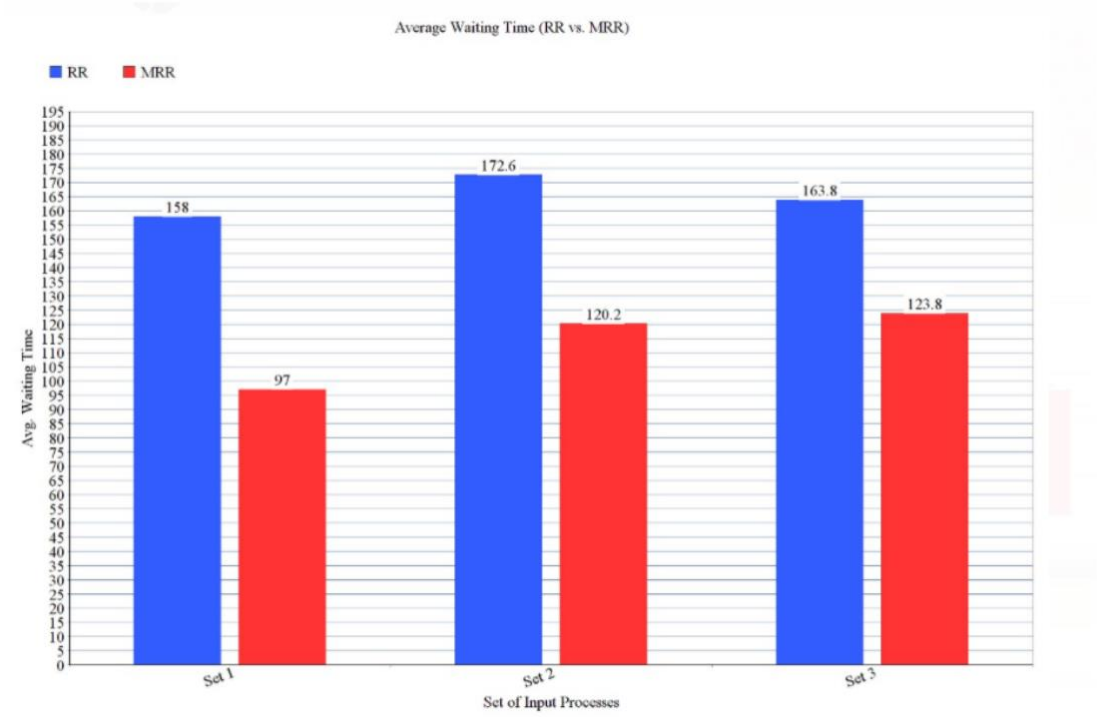
```
CPU Scheduling Algorithms
1. FCFS (First Come First Served)
2. RR (Round Robin)
3. SJF (Shortest Job First)
4. Modified Algorithm
5. Exit
Enter your choice:1
Enter the number of processes: 5
Enter the burst time of process P[1]: 12
Enter the burst time of process P[2]: 13
Enter the burst time of process P[3]: 14
Enter the burst time of process P[4]: 15
Enter the burst time of process P[5]: 16

Process Scheduling Order: P[1], P[2], P[3], P[4], P[5],

Process_id      waiting Time    Turnaround Time
 P[1]           0               12
 P[2]           12              25
 P[3]           25              39
 P[4]           39              54
 P[5]           54              70
Average Waiting Time: 26.00
Average Turnaround Time: 40.00
Throughput: 14.000000
```

II. SJF

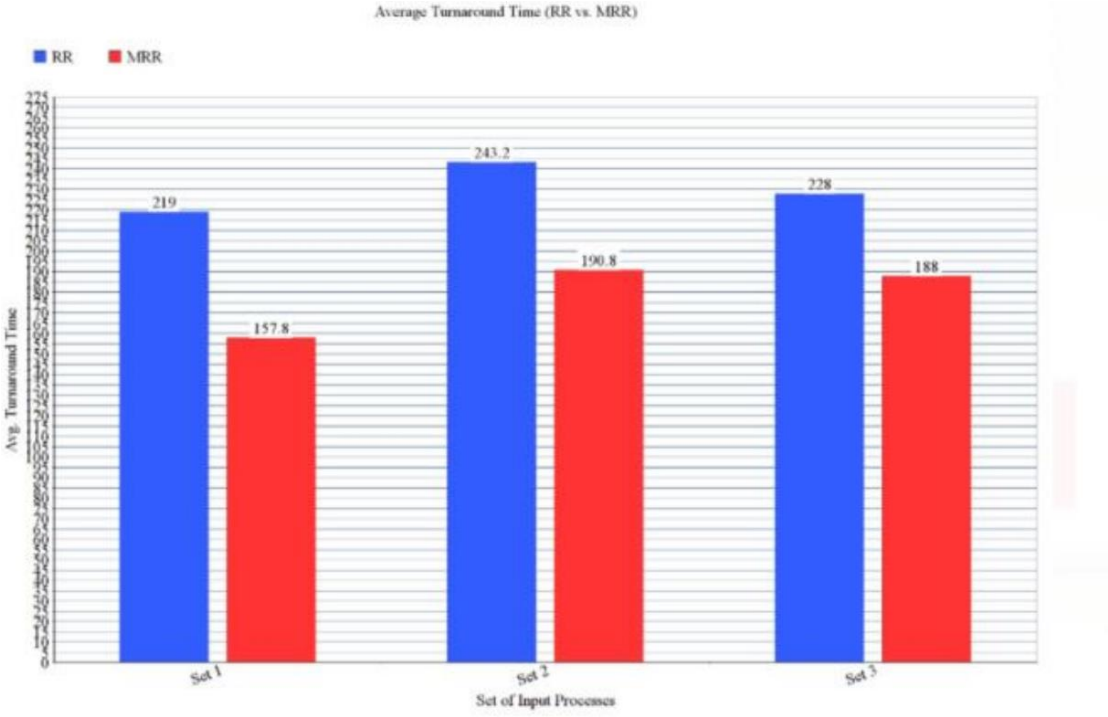
```
1. FCFS (First Come First Served)
2. RR (Round Robin)
3. SJF (Shortest Job First)
4. Modified Algorithm
5. Exit
Enter your choice:3
Enter the number of processes: 5
Enter the burst time of process P[1]: 12
Enter the burst time of process P[2]: 13
Enter the burst time of process P[3]: 14
Enter the burst time of process P[4]: 15
Enter the burst time of process P[5]: 16

Process Scheduling Order: P[1], P[2], P[3], P[4], P[5],

Process_id      waiting Time    Turnaround Time
 P[1]           0               12
 P[2]           12              25
 P[3]           25              39
 P[4]           39              54
 P[5]           54              70
Average Waiting Time: 26.00
Average Turnaround Time: 40.00
Throughput: 14.000000
```

Analysis

a) Round Robin vs Modified Round Robin for Average Waiting Time



b) Round Robin vs Modified Round Robin for Average Turnaround Time



Implementation of Machine Learning

The Second step, to run the Modified Round Robin using C

Requirements: - Python 2.7/3.7

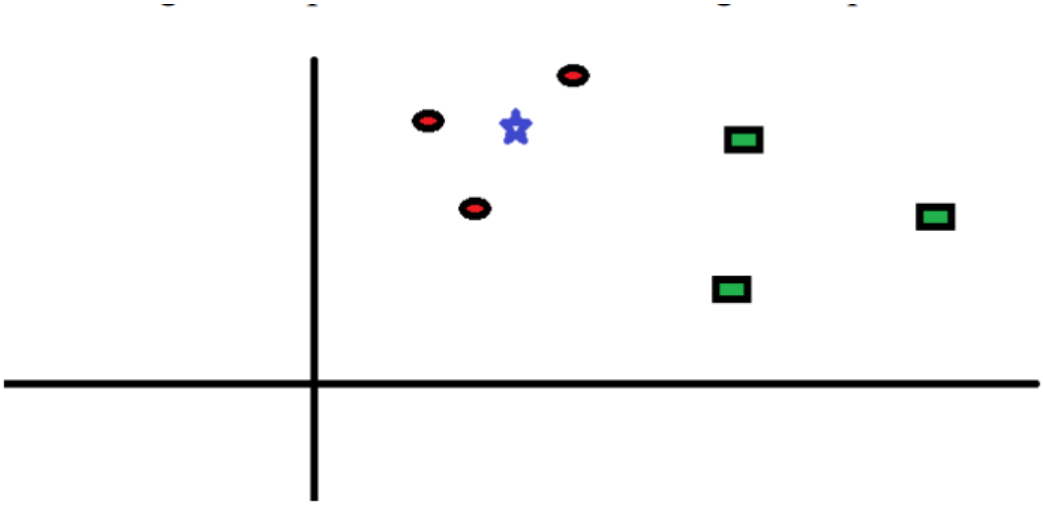
- Pandas
- Sklearn
- Matplotlib
- Numpy
- Seaborn
- Python IDLE/ Jupyter notebook

So, the implementation of CPU scheduling algorithms such as shortest job first (SJF) and shortest remaining time first (SRTF) is relying on knowing the length of the CPU burst time. So, we will propose a Machine Learning based approach on predicting the CPU burst time using Support Vector Machines and KNN algorithm.

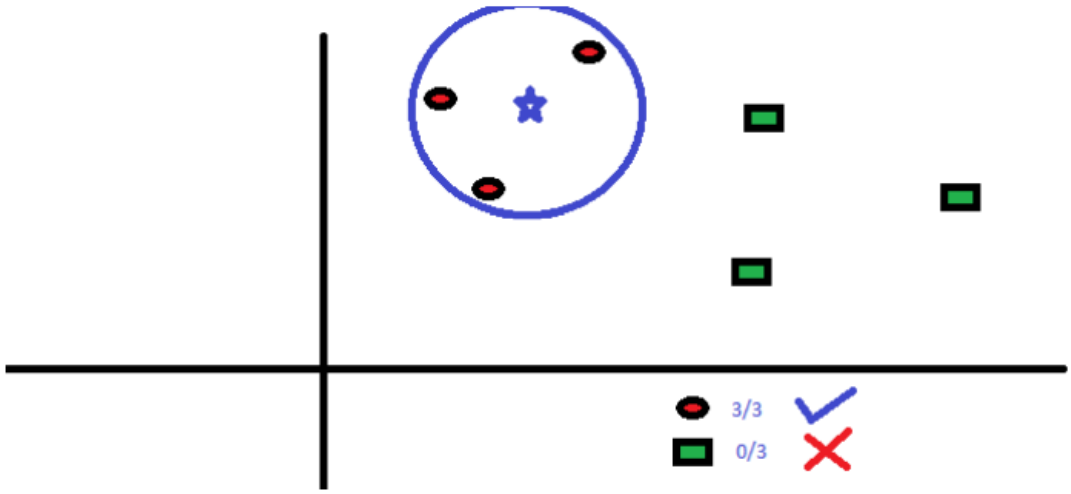
KNN Classification

Let's take a simple example to understand the algorithm.

Following is the spread of red circles and green squares.



The classification of the blue star has to be predicted. So, in KNN classification K is the nearest neighbour through which we have to take vote from. Let's say K=3 so we see the nearest 3 datapoints from the blue star and the maximum number of datapoints belong to the red circle so the blue star belongs to the red circle class.



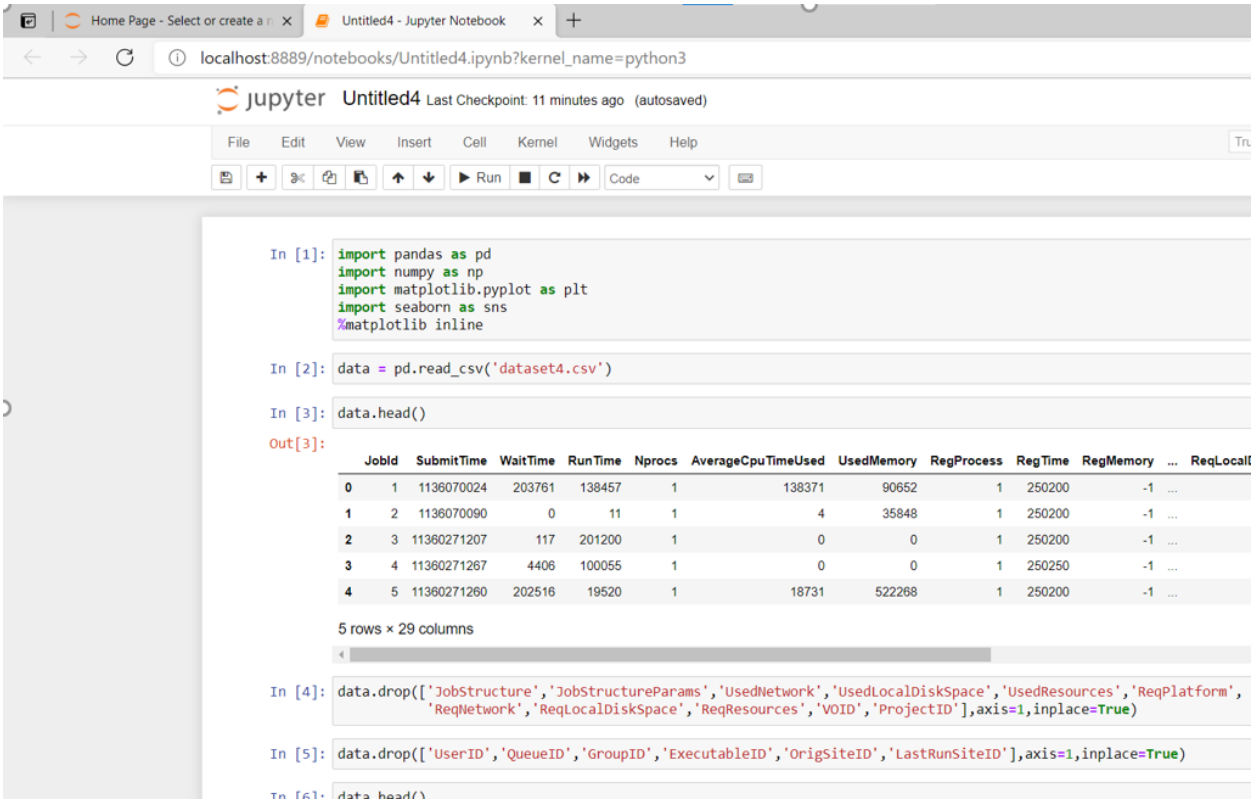
KNN Algorithm

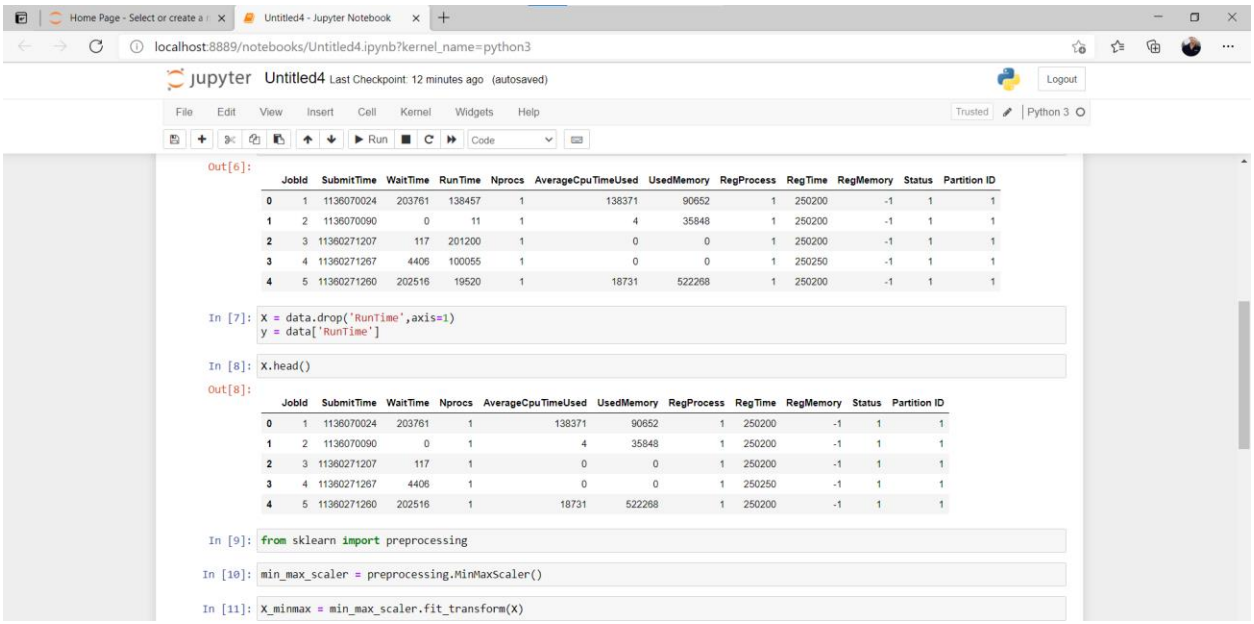
- load the data
- Initialise the value of K
- For getting the predicted class, iterate from 1 to total number of training data points.
    1) Calculate the distance between the data points and each row of training data. Here we will use the Euclidian distance metric. The other metrics can be used are Chebyshev, cosine, etc.
    2) Sort the calculated distances in ascending order based on distance values.
    3) Get top k rows from the sorted array
    4) Get the most frequent class of these rows.
    5) Return the predicted class.

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

data = pd.read_csv('processes_datasets.csv') data.head()

data.drop(['JobStructure','JobStructureParams','UsedNetwork','UsedLocalDiskSpace','UsedResources','Req
```

```
Platform','ReqN etwork', 'ReqLocalDiskSpace','ReqResources','VOID','ProjectID'],axis= 1,inplace=True)

data.drop(['UserID','QueueID','GroupID','ExecutableID','OrigSiteID','LastRunSiteID'],axis=1,inplae=True)

data.head()

X = data.drop('RunTime ',axis=1)
y = data['RunTime ']

X.head()

from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()

X_minmax = min_max_scaler.fit_transform(X)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test= train_test_split(X_minmax,y,test_size=0.30,random_state=50)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=4)

knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)

y_pred_knn = knn.predict(X_test)
y_pred_knn

from sklearn.metrics import accuracy_score, r2_score

r2_score(y_pred_knn, y_test)
```






```
C:\Users\RAHUL\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:334: DataConversionWarning: Data with input dtype int6
4 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

In [9]:
```
from sklearn import preprocessing
```

In [10]:
```
min_max_scaler = preprocessing.MinMaxScaler()
```

In [11]:
```
X_minmax = min_max_scaler.fit_transform(X)
```

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
X_train, X_test, y_train, y_test=train_test_split(X_minmax,y,test_size=0.30,random_state=50)
```

In [14]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [15]:
```
knn = KNeighborsClassifier(n_neighbors=4)
```

In [16]:
```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm'auto',leaf_size=30,metric='minkowski',
        metric_params=None, n_jobs=None, n_neighbors=4, p=2,
         weights='uniform')
```

In [17]:
```
knn.score(X_test, y_test)
```

```
0.22258418348411999
```

In [18]:
```
y_pred_knn = knn.predict(X_test)
y_pred_knn
```

```
array([20976, 39986,   14, ..., 16301,  9947, 62371], dtype=int64)
```

In [19]:
```
rom sklearn.metrics import accuracy_score, r2_score
r2_score(y_pred_knn, y_test)
```

```
0.8644730548750641
```

VI. CONCLUSION, LIMITATIONS AND SCOPE FOR FUTURE WORK

In conclusion, the modified round robin algorithm reduces the context switching, average waiting time and average turnaround time even more. So that computer system will be more efficient. The above algorithm can be useful for completing the process even faster.

The Modified Round Robin algorithm is much better than traditional round robin algorithm. But we can't ignore the fact that the result we wanted to get that even less average waiting time, turnaround time; we had not got that. So, in future, it is planned to reduce the average waiting time, turnaround time even more. And we will be working on it soon

VII. REFERENCES

1) A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average *Abbas Noon, Ali Kalakech, Seifedine Kadry* https://arxiv.org/ftp/arxiv/papers/1111/1111. 5348.pdf

2) *Sohrawordi, Ehasn Ali, Palash Uddin and Mahabub Hossain*. (2019); "A MODIFIED ROUND ROBIN CPU SCHEDULING ALGORITHM WITH DYNAMIC TIME QUANTUM". Int. J. of Adv. Res. 7 (Feb). 422- 429] (ISSN 2320-5407).

   http://www.journalijar.com/uploads/218_IJAR-26194.pdf

3) *Yosef Berhanu, Abebe Alemu and Manish Kumar Mishra*. "Dynamic Time Quantum based Round Robin CPU Scheduling Algorithm". International Journal of Computer Applications 167(13):48-55, June 2017.

   https://www.ijcaonline.org/archives/volume167/number13/berhanu-2017-ijca-914569.pdf

4) Round Robin Scheduling Algorithm based on Dynamic time quantum by *RashmiDhruv*

   https://www.ijeat.org/wpcontent/uploads/papers/v8i6/F8070088619.pdf

5) *M. U. Farooq, A. Shakoor and A. B. Siddique*, "An Efficient Dynamic Round Robin algorithm for CPU scheduling," 2017 International Conference on Communication, Computing and Digital Systems (CCODE), Islamabad, 2017, pp. 244-248,

   10.1109/C-CODE.2017.7918936.

6) International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol. 5,No.1, February 2015

   https://arxiv.org/ftp/arxiv/papers/1605/1605.00362.pdf

7) *Manish Kumar Mishra and Dr.Faizur Rashid*, AN IMPROVED ROUND ROBIN CPU SCHEDULING ALGORITHM WITH VARYING TIME QUANTUM, International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.4, No.4, August 2014 https://www.researchgate.net/profile/Faizur_Rashid/publication/273011366_An_Improved_Round_Robin_CPU_Scheduling_Algorithm_with_Varying_Time_Quantum/links/5e3bd92d

92851c7f7f201c5a/An-Improved-Round-Robin-CPUScheduling-Algorithm-with-Varying-Time-Quantum.pdf

8) *Abdulrazaq Abdulrahim, Saleh E Abdullahi, Junaidu B. Sahalu*, A New Improved Round Robin (NIRR) CPU Scheduling Algorithm. International Journal of Computer Applications (0975 – 8887) Volume 90 – No 4, March 2014

https://eduwavepool.unizwa.edu.om/lmsdatapool/00008828 /LearningObjects/paper%201.pdf