EE5903 Real Time System CA2

Variants of Round Robin CPU Scheduling Algorithm Based on Dynamic Quantum Time

Done by: Emil Yong Kai Wen

A0169907M

+65 9109 9666

Abstract

In some life-critical situations, it may require a scheduling process that is based on CPU scheduling algorithm. There are various algorithms that try to solve the problem of optimal scheduling. One of the most famous CPU scheduling is Round Robin (RR) algorithm. RR is significantly dependent on the choice of Time Quantum (TQ) used. However, classical RR uses a fixed TQ which significantly affects the performance if TQ is chosen unintelligently. This paper proposed 2 variants of RR algorithms called EDRR and IDRR. The TQs are dynamically changed based on careful calculation and evaluated on different distributions, arrival times, and sizes of datasets. After which, both algorithms are compared based on Average Waiting Time (AWT), Average Turnaround Time (ATT), starvation, Throughput, and Context Switching. It is found that IDRR has a better performance and is more robust to different variations of task set.

1) Introduction

An operating system (OS) is a system software that interacts between the user and computer hardware. Its sole purpose is to provide a platform where users can execute programs in a well-located and efficient manner. With the advancement of modern technology, OS and time-sharing systems have evolved from a single task to a multitasking environment where processes run in a synchronized manner. Therefore, CPU scheduling is one of the essential elements that an operating system must provide for multitasking [2]. It is a process of determining which process will own the CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS must ensure that it will select one of the processes available in the ready queue for execution. This selection process is performed by the CPU scheduler. Hence, different CPU scheduler algorithms will produce different results

With the current COVID-19 pandemic going on, the allocation of vaccines poses a huge challenge to populated countries such as India [1]. Naively, the current ranking system is based on the concept that health care personnel should be given first, followed by frontline, seriously ill patients, the elderly, and then the public in a First Come First Serve (FCFS) fashion. This method poses a huge problem for Vaccine Scheduling. When the second wave of infection hits India, many lives were lost even though vaccines were available. Due to the poor scheduling technique, no one was taking them as they think it was unavailable. Even if people know that registration is open to them, only a handful of citizens are given the chance to register in a small window time because of the FCFS approach [1].

With this in mind, it is essential to have an optimized CPU scheduling algorithm for critical-based situations like the vaccine scheduler for pandemics [2]. The efficiency of CPU scheduling algorithms can be evaluated based on Waiting Time (WT), Turnaround Time (TT), Context Switching (CS), Response Time (RT). TT is the total amount of time spent by the process from coming in the ready state for the first time to its completion. WT on the other hand is the total time the process has to wait for the CPU before it can execute. CS is a process involving switching the CPU from one process to another. In this phenomenon, the execution of the process that is presenting the running state is suspended by the kernel, and another process that is present in the ready queue will take over and execute by the CPU. Response time is the time spent when

the process is in the ready state and gets the CPU for the first time. Lastly, Throughput. It defines the number of processes execute by the CPU in a given time Since each process can have different values, a unified solution is to calculate the average for comparison between different schedulers. The calculation for average waiting time (AWT), turnaround time (ATT) and response time (ART) are as follows[2][3]:

$Waiting\ time, WT_i = Turnaround\ time - Burst\ time$ $\textbf{Avg Waiting Time (AWT)} = \{\sum \mathbf{W_{T_i}}\}/\mathbf{n} \quad (1)$	$Turnaround\ Time, TT_i = Turnaround\ time - Burst\ time$ $\textbf{Avg Turn around Time (ATT)} = \{\sum \mathbf{T_{T_i}}\}/\mathbf{n} \quad (2)$
$Response\ Time, RT_i = Time\ at\ which\ process\ gets\ the\ CPU\ for\ the\ first\ time - Arrival\ time$ $\textbf{Avg Response Time (ART)} = \{\sum \boldsymbol{R_{T_i}}\}/\mathbf{n} \quad (3)$	

In this paper, the two main algorithms to be evaluated are Efficient Dynamic Round Robin Algorithm **(**EDRR) and Improved Dynamic Round Robin (IDRR) which is based on a different methodology to calculate Time Quantum (TQ) and schdeuling. Both algorithms are variants of the classical Round Robin (RR) algorithm and are said to achieve better performance metric than RR. This paper will analyze and compare the performance based on the above-mentioned performance metrics with a different distribution, size, and arrival times of datasets. At the end of the paper, this paper will compare the effectiveness and disadvantages of the two algorithms.

Problem with classical Round Robin

Round Robin (RR) scheduling is a preemptive algorithm that is heavily dependent on the quantum time to determine its performance. It generally employs time-sharing by giving each process a time slot or quantum for the CPU. However, if the time quantum is set too small, it will incur a larger overhead for context switching which will result in increased waiting time for the process. Meanwhile, if the quantum time is set too high, RR will behave like the FCFS algorithm which defeats the purpose of implementation.[2] Therefore, the efficiency of RR greatly depends on the ability to choose the correct time quantum to maximize the efficiency. In classical RR, the time quantum is constant throughout. Sometimes, with an unintelligent choice of time quantum, some processes must wait for all other processes to complete even if their remaining execution time is much smaller.

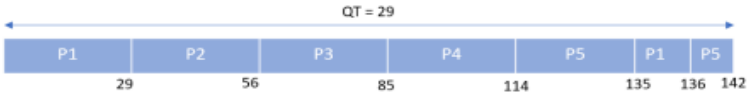


Figure 1: Problem with classical RR (Gannt chart)

As shown in Figure 1, processes P1 and P5 have a remainder of 1 or 2 units of time. But because of the rigid quantum time, P1 and P5 must wait for all other processes to complete before they can execute the remaining time. This effect may snowball and substantially cause an unnecessary increase in waiting time and turnaround time. Therefore, this sparks an initiative to have a dynamic quantum time that changes efficiently – EDRR and IDRR CPU algorithm.

2) Efficient Dynamic Round Robin Algorithm (EDRR)

As aforementioned, it is redundant to leave processes waiting if they only have a small amount of execution time left. EDRR [2] is proposed to tackle this issue where it provides an elastic dynamic

time quantum that will allow a process to execute completely if its remaining execution time is lesser than or equal to 0.2th of the initial burst time [2]. This is said to ensure higher efficiency, lesser context switches, average waiting time, and turn around time with higher throughput since processes have a lesser probability of remaining with just 1 or 2 execution time

EDRR[2] first finds the maximum burst time available in the ready queue. In EDRR, the time quantum is dynamic based on percentage. The time quantum is set as 0.8th of the maximum burst time found in the ready queue. This ensures that most of the processes have adequate time to execute. The scheduler then allocates the CPU to all processes in the ready queue that have a burst time lesser than the time quantum. The process with a larger burst time, more than 0.8th of the maximum time quantum **will be placed on hold and wait in the ready queue** while waiting for smaller processes to execute finish. Once completed, the time quantum is recalculated to the maximum burst time of the remaining process. In the case of different arrival time, the scheduler will readjust the time quantum if a new process reached the ready queue at the end of the current time quantum. This ensures that all processes will have enough time to complete all task execution.

3) Improved Dynamic Round Robin (IDRR) Scheduling Algorithm

IDRR [3] is another variant of RR that calculates time quantum in another way that ensure more processes to finish their execution time. IDRR time quantum is dynamically changed at different specific rounds of iteration. It will ensure that the TQ is not set as too large else, behaving like FCFS[3]. For same arrival task, the processes in the ready queue are first sorted out in ascending order according to the burst time. In the first round, the TQ is set by taking the average of the 2 largest burst times. After which, for the remaining task, the TQ is updated to the average of the largest remaining task execution time with the previous TQ used. Whereas for different arrival task, IDRR will look at the available task in the ready queue and set the TQ to the maximum burst time. When more processes arrive in the ready queue, the first update to the TQ is calculated by the average of the largest remaining burst time and the previous TQ minus the average of the 2 smallest remaining arrival time. Subsequently, in the future TQ updates, if there are more tasks available, TQ is updated by subtracting the average of the highest remaining burst time with the previous quantum time and half of the lowest remaining arrival time. Thus, IDRR is said to ensure TQ to be as large as possible but not behaving like FCFS to improve the performance

4) Description of Input Data

In this report, we will be evaluating 6 different datasets generated based on two variations of probability distribution that describe all possible values and the likelihood that a random variable can take with a given range. Of the 6, 4 datasets are based on different arrival time while the other 2 are with same arrival time. This is to simulate real-life scenario of tasks arriving at the CPU. The first distribution is the Binomial distribution. The binomial distribution is one of the discrete probability distributions used when there are mutually exclusive outcomes of a trial. Thus, a binomial distribution has the following properties 1) A binomial experiment consists of n repeated trials where each trial can result in just two outcomes: success or failure. 2) The probability of success is the same for each outcome and hence all trials are independent which is

crucial for the dataset. Mathematically, $B(x;n,p) = \frac{n!}{x!(n-x)!}P^x(1-P)^{n-x}$ where x is the number of successes in each trial.

The second distribution is normal Gaussian distribution, a common distribution function for independent, randomly generated variables that are parameterized by sigma (variance) and mean. Using the parameters as shown in the tables below. Each dataset generates a different number of tasks. The number of tasks sets for gaussian are 30,150,350,550,850,1000. Whereas the number of task set generated by binomial starts from 850, 1000, 1500 and 2000.

Different arrival time			
Gaussian	Dataset 1	Arrival Time: μ =10 , σ = 5	Burst Time: μ =50 , σ = 20
Gaussian	Dataset 2	Arrival Time: μ =10 , σ = 5	Burst Time: μ =50 , σ = 35
Gaussian	Dataset 3	Arrival Time: μ =10 , σ = 10	Burst Time: μ =50 , σ = 20
Binomial	Dataset 4	Arrival Time: n =15 , p = 0.5	Burst Time: n =100 , p = 0.4

Table 1 & 2 : Different Arrival and Same Arrival time parameterized differently

Same arrival time		
Gaussian	Dataset 5	Burst Time: μ =50 σ = 20
Gaussian	Dataset 6	Burst Time: μ =50 σ = 35

In addition, to utilize the generated dataset, there are a few assumptions made alongside the task generation.

Assumption 1	For same arrival task, the assumption is that all process arrives at t = 0.
Assumption 2	All burst time must be larger than 0
Assumption 3	For different arrival time, there must not be negative arrival time and some process must be in the ready queue at t = 0
Assumption 4	For different arrival time, CPU must not become IDLE between 2 given process. Since task set are generated randomly with different parameters, arrival time between two processes might be far apart in different datasets and this will influence the accuracy on the effectiveness of the algorithm. Hence generated task set must not have idle arrival time between process
Assumption 5	For IDRR, in different arrival time, it is assumed that at least 2 new tasks is in the ready queue after first iteration of updating quantum time

5 Experimental Results

5.1 Analyses of same arrival time using Dataset 5 with only 30 generated tasks.

The first analysis in this section uses Gaussian Dataset 5's parameters with 30 generated task sets arriving at the ready queue at t = 0. This result is simulated using the dataset filename " simulation_samearrival_30_bt_50,20". Figures 2, 4, and 6 show the performance metric of IDRR whereas Figures 3,5, and 7 are the performance metric of EDRR using the same dataset.

Comparing the different metrics in Figures 2 and 3, it is obvious that IDRR's performance is better than EDRR. Even though both algorithms have the same number of context switches, the average waiting time for IDRR is 595ms as compared to 676.4ms in EDRR. ATT for IDRR is also better with 648.5ms vs 729.13ms in EDRR. Similarly, IDRR response time is also faster than EDRR. From the Gantt charts between Figures 6 and 7, we can see that IDRR allows larger tasks (eg.22) to execute whereas, in EDRR, larger tasks have to wait at the end to execute.

```
Average Turnaround Time (ATT): 648.5
Average Waiting Time (AWT): 595.7666666666667
Context Switching (CS): 29
Average Response Time (ART): 595.7666666666667
```

Figure 2: IDRR Performance Metric

```
Average Turnaround Time (ATT): 729.1333333333333
Average Waiting Time (AWT): 676.4
Context Switching (CS): 29
Average Response Time (ART): 676.4
```

Figure 3: EDRR Performance Metric

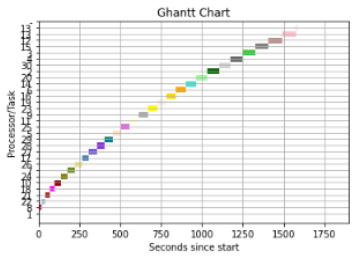


Figure 4: Plotted Gantt Chart (IDRR)

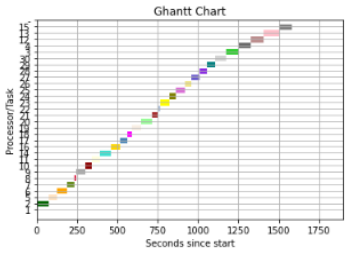


Figure 5: Plotted Gantt Chart (EDRR)

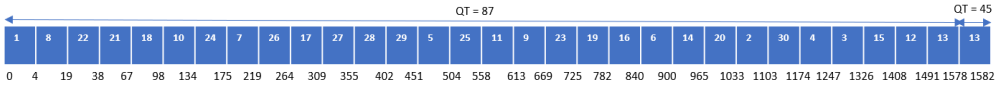


Figure 6: Detailed Gantt Chart (IDRR)

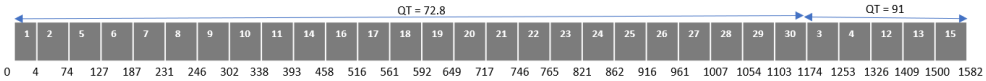


Figure 7: Detailed Gannt Chart (EDRR)

5.2 Analyses of different arrival times using Dataset 1 with 30 generated tasks

However, it is too quick to conclude that IDRR is better. This section evaluates the performance by having task sets arriving at different arrival times with the same distribution parameters for burst time(Dataset 1). The task set file name is “Simulation_diffarival_30_at10_5_bt50_20”.

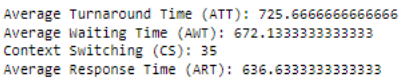


Figure 8: IDRR Performance Metric

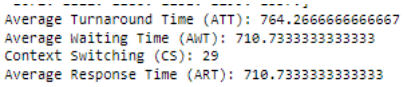


Figure 9: EDRR Performance Metric

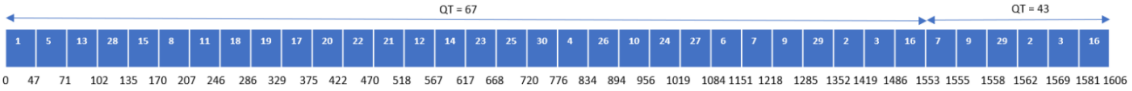


Figure 10: Detailed Gantt Chart (IDRR)

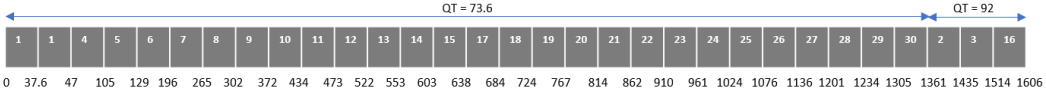


Figure 11: Detailed Gannt Chart (EDRR)

Figure 8 and Figure 9 reinforced the idea that IDRR is indeed superior. AWT is 672ms for IDRR while AWT for EDRR is 710.73. Furthermore, ATT and ART are faster in IDRR than EDRR. Even though IDRR context switching is more than EDRR by 6, CS time is usually kept very low in most systems. If a constant CS of 0.5 is considered, IDRR still outperforms EDRR. The reason for higher CS is mainly because IDRR calculates and ensures smaller TQ as compared to EDRR.

5.3 Analyses of same arrival time using larger dataset and different parameters (Gaussian).

Even though Section 5.1 shows that IDRR is superior to EDRR, a small dataset of 30 cannot fully justify the performance, in this section, the analyses will evaluate a larger number of task sets (eg.150,550,850,1000) and with different gaussian parameters for the burst time as shown in Table 1. The difference in parameters is primarily varying the standard deviation.

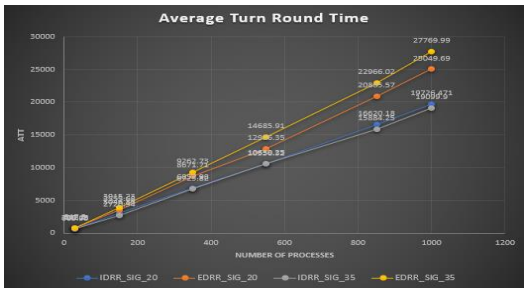


Figure 12: ATT (Same Arrival)

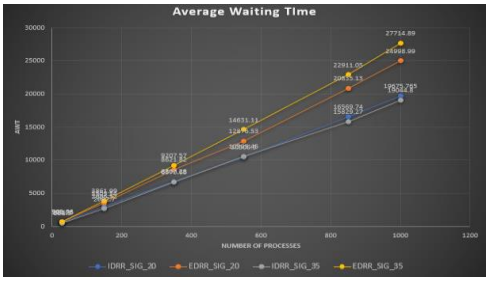


Figure 13: AWT (Same Arrival)

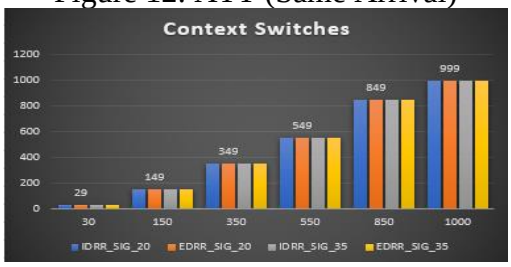


Figure 14: CS (Same Arrival)

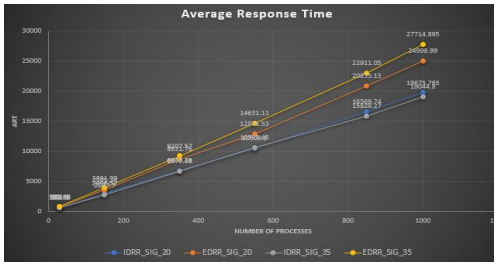


Figure 15: ART (Same Arrival)

As shown in Figures 12 to 15, the grey and yellow plots correspond to IDDR and EDRR respectively with a gaussian distribution using sigma 35 instead of 20 on different sizes of data set as aforementioned. From the plots, the number of context switches does not vary much and is constant throughout the experiment with different sigma and number of data sizes. However, these figures still show that IDRR is more efficient than EDRR. In both sigma distributions (20 & 35) datasets across different data sizes, IDRR's AWT, ATT, and ART are consistently lower than EDRR as shown by the gray and blue lines. This shows that IDRR is still superior even in a larger dataset. In addition, if you compare EDDR's graphs (yellow and orange) versus IDRR's graph (gray and blue), the plots show that as the data size increase, it starts **fanning out** and **diverging**.

This is a strong indicator that as the number of data size increases, EDRR performance is getting worse (higher) as compared to IDRR. Furthermore, even with different sigma values, the performance of IDRR is quite relative to each other, unlike EDRR. Both gray and blue plots are almost overlapping each other. However, at sigma 35, EDRR's performance is worse as compared to sigma 20. Increasing standard deviation means increasing the variability in the dataset where each value lies further away from the mean. This shows that IDRR can perform relatively well even in such a scenario as compared to EDRR. This is desired since real tasks can be quite distributed.

5.4 Analyze different arrival times using larger dataset and different parameters (Gaussian).

In order to reinforce section 5.3 understanding, section 5.4 will evaluate with more variability on the parameters and dataset used. Similarly, this section will evaluate larger datasets but with different arrival time. Figure 16 to 19 shows experimental results of data set sizes of different arrival time with parameters shown in Table 1. The blue and orange plots are IDRR and EDRR respectively with an arrival time mean of 10, sigma 5; burst time mean of 50, sigma 20. Gray and yellow plots are with similar arrival time gaussian parameters as blue and orange plots but burst time with mean 50, sigma 35 for the distribution. On the other hand, light blue and green plots differs in arrival time with a mean of 10 and sigma 10. Burst time mean is kept at 50 and sigma 20

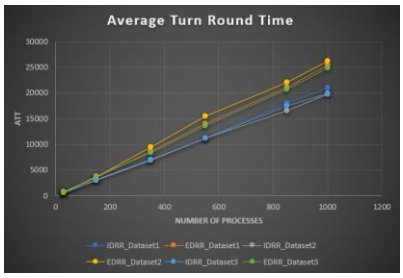


Figure 16: ATT (Different Arrival)

Figure 17: AWT (Different Arrival)

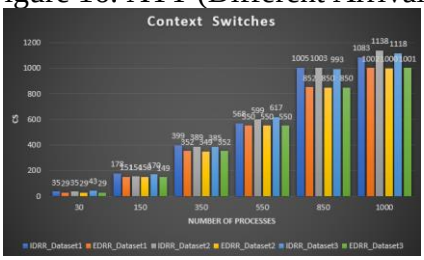


Figure 18: CS (Different Arrival)

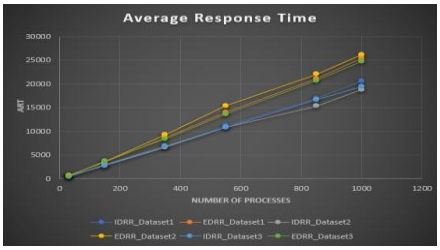


Figure 19: ART (Different Arrival)

These figures show that for all 3 variations of parameters, IDRR (Blue, light blue, and gray)'s performance metric, AWT, ATT, and ART are still better than EDRR.

5.5 Analyze standard deviation among different datasets

Among the different variations of the standard deviation of datasets, dataset 2 outperforms the rest of the datasets for different arrival times task sets and dataset 6 outperforms dataset 5 for the same arrival time gaussian distribution task sets. The similarity between datasets 2 and 6 is the standard deviation. Both datasets have the **highest** standard deviation burst time in their respective arrival time. In addition, with increasing standard deviation and datasets size, the **fanning** and **diverging** phenomenon are more obvious. This implies that IDRR's performance will get better than EDRR as they increase. Unfortunately, context switching for IDRR in Figure 18 at 1000 task set for dataset 2 is still more than EDRR; 1138 vs 1000 respectively. This is because IDRR handles the issue of **starvation** by giving the larger task a chance and setting the TQ to as low as possible. IDRR's TQ will always be smaller than EDRR since EDRR set TQ to the largest burst time after all smallest task has been executed while IDRR uses averaging. Thus, EDRR is guaranteed to have a smaller number of CS as compared to IDRR. Even though the context switches are larger in IDRR but if a constant CS time of 0.5 is considered, IDRR still outperforms EDRR for any given data size.

Figure 20 and 21 shows the throughput performance of various datasets. Both figures consider throughput without factoring in the CS time since CS time is dependent upon hardware support like waiting for input-output and reading etc. In Figure 20, the throughput for EDRR and IDRR in different arrival time datasets across different standard deviations is quite similar. Similarly, for the same arrival time taskset, the throughput is quite similar as well. Taking 1000 taskset as an example in Figure 21, the difference between the two different task sets is only 0.16% difference. However, as aforementioned, IDRR has a larger number of CS as compared to EDRR. A larger CS count means total execution time for IDRR will always be longer than EDRR if both algorithms considers a fixed CS time. However, from Figure 18, the CS difference between IDRR and EDRR is not drastically different. If CS is kept low and throughput performance is not that of

importance, and considering other performance metrics, IDRR is still superior to EDRR.

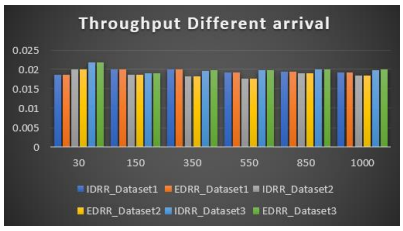


Figure 20: Throughput Different Arrival

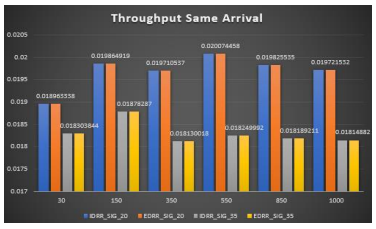


Figure 21: Throughput Same Arrival

5.6 Analyze with Binomial distribution

The above phenomenon is reinforced with binomial distribution with **even** larger dataset (1000 ~ 2000). IDRR's still has a better performance metric as compared to EDRR with better AWT, ART and ATT but larger CS. Refer to appendix to see more graph results the with binomial distribution.

6 Comparison of EDRR and IDRR with pitfalls and strengths

Overall, IDRR shows better schedulability than EDRR. IDRR achieves better AWT, ATT and ART in a larger and variable task set as compared to EDRR. IDRR also ensures that the Time Quantum is dynamically chosen appropriately to prevent large CS or degeneration to FCFS algorithm. Furthermore, IDRR addresses the issues of starvation. Starvation can be defined as a process being denied access to the CPU for execution. IDRR still allocates CPU to large tasks to prevent longer AWT and ATT with proper calculation of TQ. Unfortunately, this leads to a larger number of CS as compared to EDRR. In contrast, starvation was not addressed in EDRR. Large tasks have to wait till smaller tasks are finished. This explains the longer AWT and ATT compared to IDRR in a large dataset. In addition, this shortcoming will worsen in EDRR when the dataset is extremely undistributed; a large dataset with one or two tasks that has an extremely large burst time compared to the rest. Since EDRR's TQ is percentage-based, AWT and ATT will be immensely affected since multiple small tasks exist. However, in terms of CS, EDRR algorithm performs better. If CS time is long, then EDRR will perform better than IDRR. But nevertheless, both algorithms still perform better than classical round-robin with fixed quantum time.

7 Conclusion.

This paper discussed the 2 variants of Round Robin scheduling algorithms, EDRR and IDRR. It compares the two algorithms on various aspects such as AWT, ATT, ART, starvation, context switching, and throughput on various parameters distribution and arrival time datasets. In terms of context switching, EDRR outperforms IDRR because of its larger quantum time. This was reiterated according to [2], the authors in the paper explicitly mentioned that EDRR performs well in CS. However, it claims that EDRR is the most efficient scheduling algorithm with lowest AWT and ATT. This claim is invalid because EDRR does not address the problem of starvation. Compared to IDRR which addresses starvation, IDRR's AWT, ATT, and ART are performing better than IDRR. In conclusion, the IDRR scheduling algorithm is significantly superior to EDRR in a larger dataset with low CS time. In addition, it is more adaptive to a more distributed dataset.

Reference

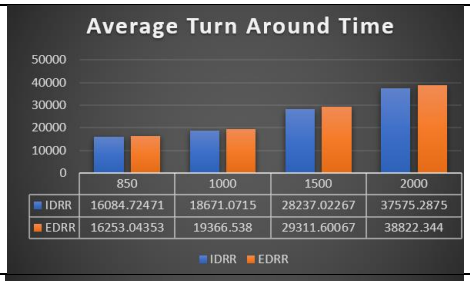
[1] V. Gondaliya, S. Patel, J. Hemnani and S. Patel, "Efficient vaccine scheduler based on CPU scheduling algorithms," 2021 International Conference on Artificial Intelligence and Machine Vision (AIMV), 2021, pp. 1-6, doi: 10.1109/AIMV53313.2021.9670986.

[2] 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), 2017, pp. 244-248, doi: 10.1109/CCODE.2017.7918936

[3] A. Alsheikhy, R. Ammar and R. Elfouly, "An improved dynamic Round Robin scheduling algorithm based on a variant quantum time," 2015 11th International Computer Engineering Conference (ICENCO), 2015, pp. 98-104, doi: 10.1109/ICENCO.2015.7416332.

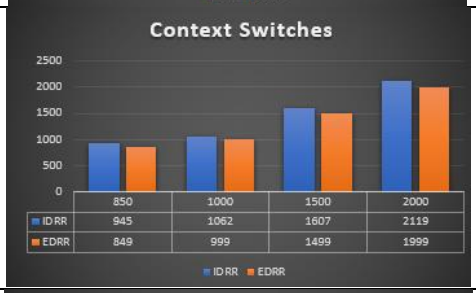
Appendix

Binomial Distribution graph for different arrival times



Tasks #	IDRR	EDRR
850	16084.72	16253.04
1000	18671.07	19366.54
1500	28237.02	29311.6
2000	37575.29	38822.34

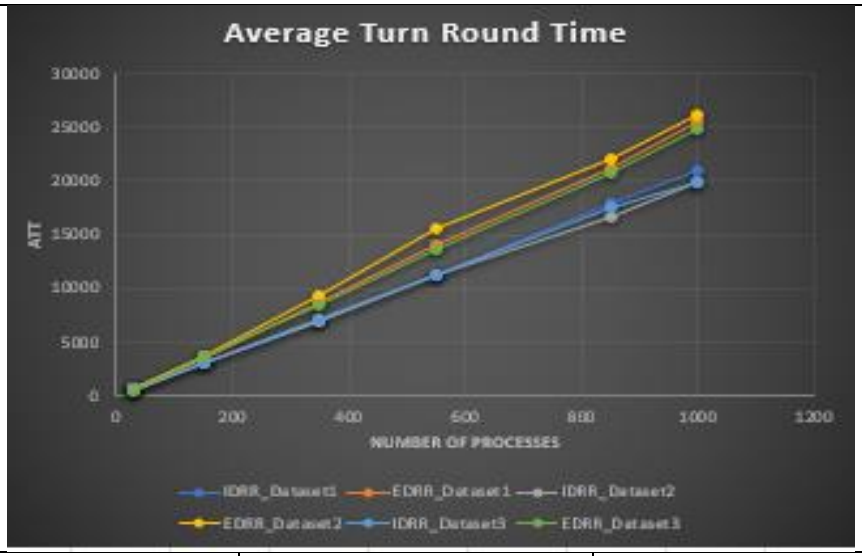
Task #	IDRR	EDRR
\T_n 15 , p 0.5	Bt_mu: n 100, p:0.	
850	16044.79	16213.11
1000	18631.11	19326.58
1500	28196.84	29271.42
2000	37535.18	38782.24



Task #	IDRR	EDRR
\T_n 15 , p 0.5	Bt_mu: n 100, p:0.	
850	945	849
1000	1062	999
1500	1607	1499
2000	2119	1999

Task #	IDRR	EDRR
\T_n 15 , p 0.5	Bt_mu: n 100, p:0.	
850	15792.19	16213.11
1000	18536.51	19326.58
1500	28009.17	29271.42
2000	37360.99	38782.24

Different Arrival time with Gaussian Distribution



Arrival Time
Mean:10 Sigma: 5

Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
AT_mu 10 , sg: 5 Bt_mu: 50, sg:20		
30	725.66	764.26
150	3017.56	3534.2
350	7024.78	8678.31
550	11255.8	14046.14
850	17980.96	21156.07
1000	20925.01	25422.85

Arrival Time
Mean:10 Sigma: 5

Burst Time
Mean: 50
Sigma: 35

	IDRR	EDRR
AT_mu 10 , sg: 5 Bt_mu: 50, sg:35		
30	559.96	685.03
150	2955.52	3729.56
350	6949.4	9393.57
550	11155.29	15466.17
850	16618.46	22115.02
1000	19768.71	26196.81

Arrival Time
Mean:10 Sigma: 10

Burst Time
Mean: 50
Sigma: 20

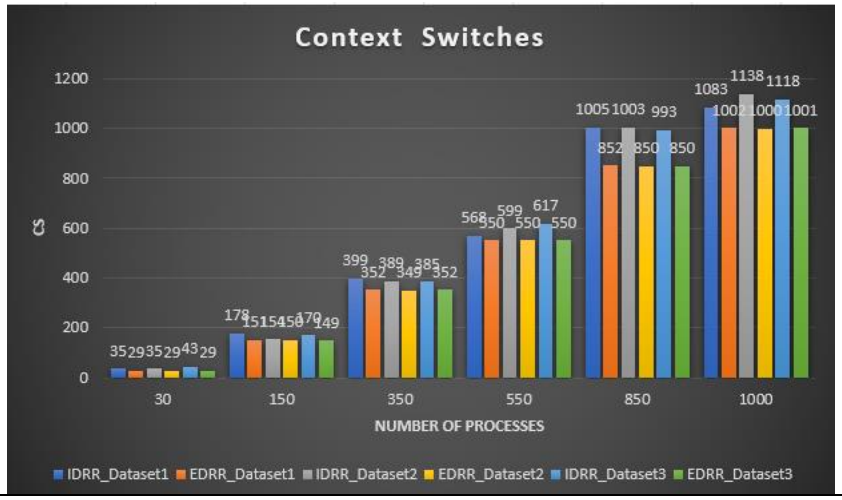
	IDRR	EDRR
AT_mu 10 , sg: 10 Bt_mu: 50, sg:20		
30	613.13	569.86
150	3157.7	3763.69
350	7098.96	8445.53
550	11217.82	13665.68
850	17549.92	20748.85
1000	19966.22	24827.12

Arrival Time Mean:10 Sigma: 5 Burst Time Mean: 50 Sigma: 20	Arrival Time Mean:10 Sigma: 5 Burst Time Mean: 50 Sigma: 35	Arrival Time Mean:10 Sigma: 10 Burst Time Mean: 50 Sigma: 20

No Task	IDRR AWT	EDRR
30	672.13	710.73
150	2967.46	3484.1
350	6974.9	8628.42
550	11203.95	13994.29
850	17929.62	21104.74
1000	20872.96	25370.8

	IDRR	EDRR
30	510.03	635.1
150	2901.96	3676
350	6894.58	9338.74
550	11098.48	15409.36
850	16565.63	22062.2
1000	19714.43	26142.53

	IDRR	EDRR
30	567.36	524.1
150	3105.31	3711.3
350	7048.56	8395.12
550	11167.32	13615.18
850	17499.82	20698.75
1000	19916.02	24776.91

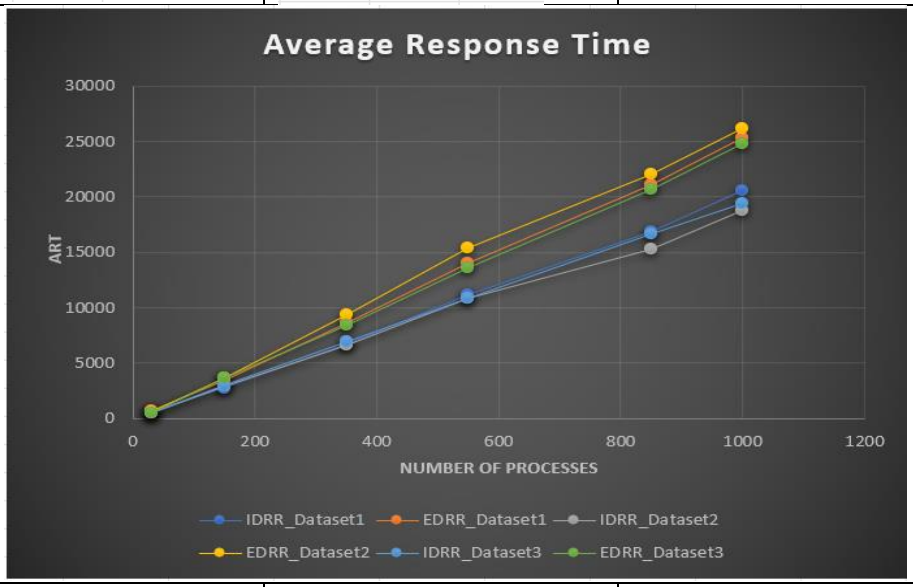


Arrival Time Mean:10 Sigma: 5 Burst Time Mean: 50 Sigma: 20	Arrival Time Mean:10 Sigma: 5 Burst Time Mean: 50 Sigma: 35	Arrival Time Mean:10 Sigma: 10 Burst Time Mean: 50 Sigma: 20

	IDRR	EDRR
30	35	29
150	178	151
350	399	352
550	568	550
850	1005	852
1000	1083	1002

	IDRR	EDRR
30	35	29
150	154	150
350	389	349
550	599	550
850	1003	850
1000	1138	1000

	IDRR	EDRR
30	43	29
150	170	149
350	385	352
550	617	550
850	993	850
1000	1118	1001



Arrival Time
Mean:10 Sigma: 5

Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	636.63	710.73
150	2785.75	3483.82
350	6723.72	8627.91
550	11170.1	13994.2
850	16905.6	21103.3
1000	20614.1	25370.4

Arrival Time
Mean:10 Sigma: 5

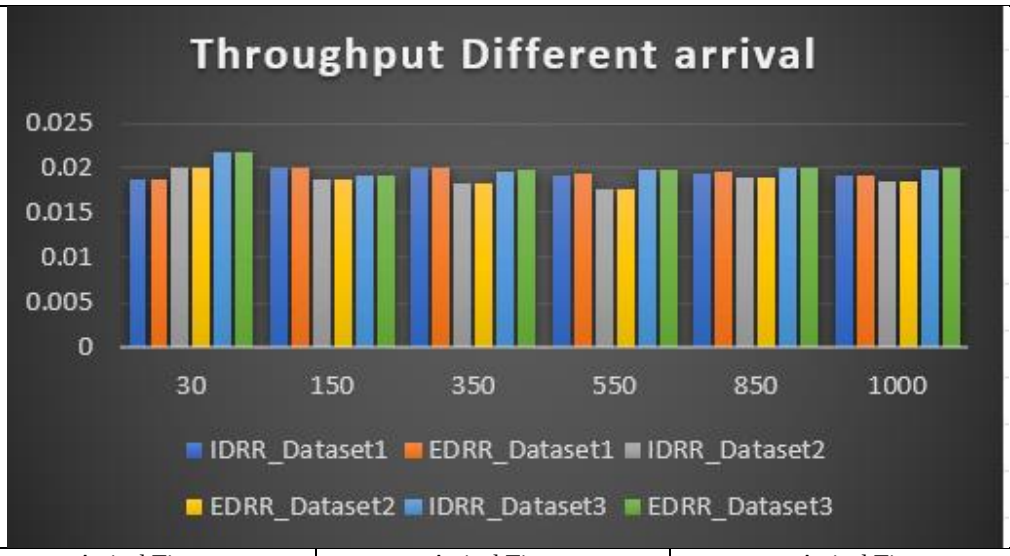
Burst Time
Mean: 50
Sigma: 35

	IDRR	EDRR
30	467.7	635.1
150	2894.69	3675.82
350	6670.59	9338.74
550	10892.7	15409.2
850	15345.2	22062
1000	18842.2	26142.5

Arrival Time
Mean:10 Sigma: 10

Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	470.56	524.1
150	2990.67	3711.3
350	6927.61	8393.07
550	10887.9	13614.82
850	16637	20697.87
1000	19406.8	24775.75



Arrival Time
Mean:10 Sigma: 5

Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	0.01868	0.01868
150	0.01995	0.01996
350	0.02005	0.02005
550	0.01925	0.01929
850	0.01947	0.01948
1000	0.01918	0.01921

Arrival Time
Mean:10 Sigma: 5

Burst Time
Mean: 50
Sigma: 35

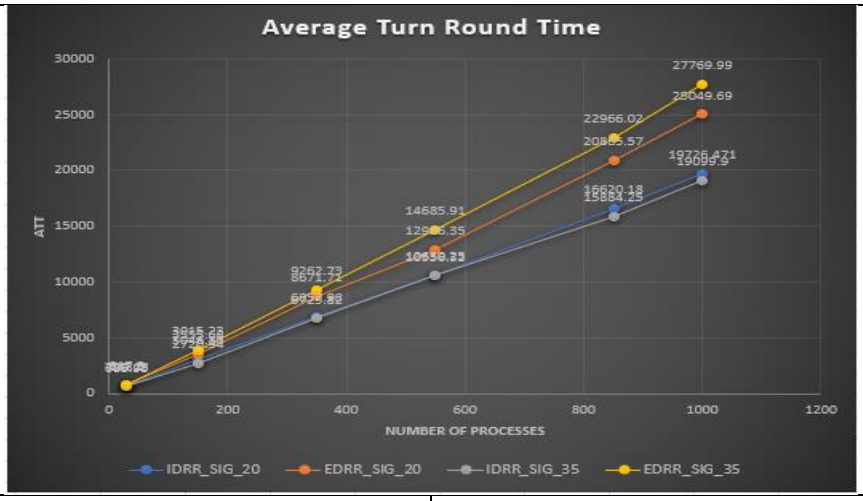
	IDRR	EDRR
30	0.02003	0.02003
150	0.01861	0.01867
350	0.01824	0.01824
550	0.01757	0.0176
850	0.01893	0.01893
1000	0.01841	0.01842

Arrival Time
Mean:10 Sigma: 10

Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	0.02185	0.02185
150	0.01909	0.0190864
350	0.01968	0.019839
550	0.01974	0.0197998
850	0.01996	0.0199615
1000	0.01987	0.0199167

Same arrival time (Gaussian)

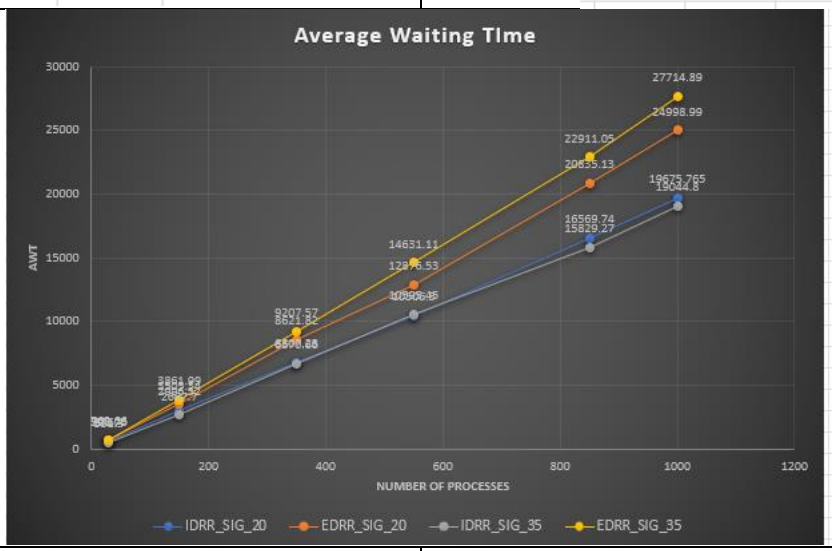


Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	648.5	729.13
150	3046.86	3555.68
350	6857.98	8671.71
550	10556.32	12926.35
850	16620.18	20885.57
1000	19726.47	25049.69

Burst Time
Mean: 50
Sigma: 35

	IDRR	EDRR
Bt_mu: 50, sg:35		
30	605.93	817.3
150	2720.94	3915.23
350	6725.82	9262.73
550	10650.25	14685.91
850	15884.25	22966.02
1000	19099.9	27769.99



Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	595.76	676.4
150	2996.52	3505.34
350	6807.25	8621.82
550	10506.5	12876.53
850	16569.74	20835.13
1000	19675.77	24998.99

Burst Time
Mean: 50
Sigma: 35

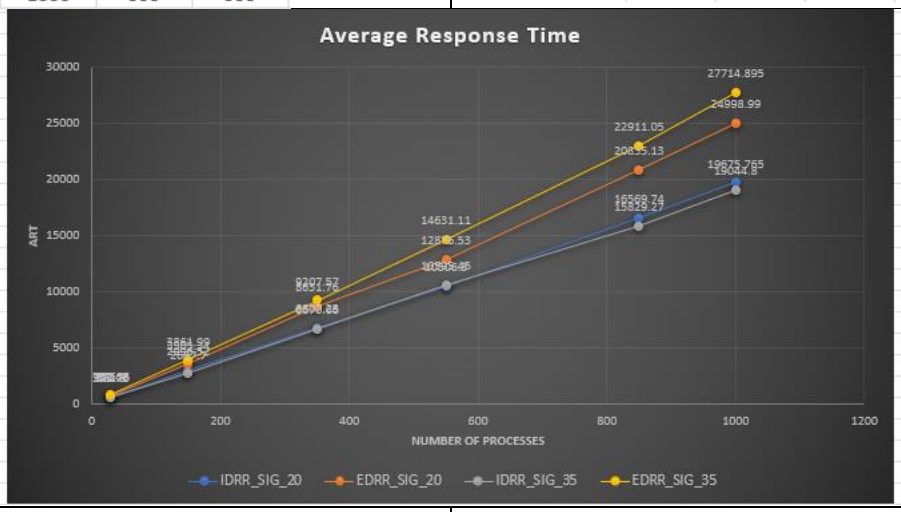
	IDRR	EDRR
Bt_mu: 50, sg:35		
30	551.3	762.66
150	2667.7	3861.99
350	6670.66	9207.57
550	10595.45	14631.11
850	15829.27	22911.05
1000	19044.8	27714.89

Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	29	29
150	149	149
350	349	349
550	549	549
850	849	849
1000	999	999

Burst Time
Mean: 50
Sigma: 35

	IDRR	EDRR
Bt_mu: 50, sg:35		
30	29	29
150	149	149
350	349	349
550	549	549
850	849	849
1000	999	999

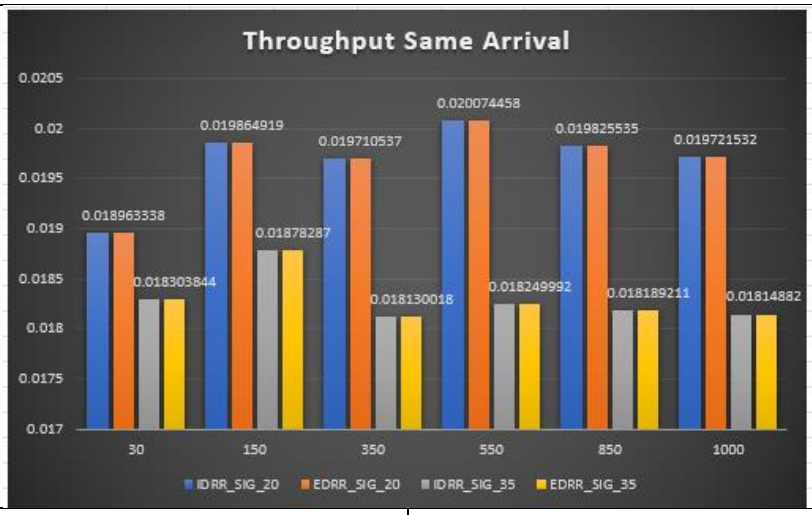


Burst Time
Mean: 50
Sigma: 20

	IDRR	EDRR
30	595.76	676.4
150	2996.52	3505.34
350	6807.25	8631.76
550	10506.5	12876.53
850	16569.74	20835.13
1000	19675.77	24998.99

Burst Time
Mean: 50
Sigma: 35

	IDRR	EDRR
AT_mu 10 , sg: 5 Bt_mu: 50, sg:35		
30	551.3	762.66
150	2667.7	3861.99
350	6670.66	9207.57
550	10595.45	14631.11
850	15829.27	22911.05
1000	19044.8	27714.9



Burst Time Mean: 50 Sigma: 20	Burst Time Mean: 50 Sigma: 35

	IDRR	EDRR
30	0.018963	0.018963
150	0.019865	0.019865
350	0.019711	0.019711
550	0.020074	0.020074
850	0.019826	0.019826
1000	0.019722	0.019722

	IDRR	EDRR
	AT_mu 10 , sg: 5 Bt_	
30	0.018304	0.018304
150	0.018783	0.018783
350	0.01813	0.01813
550	0.01825	0.01825
850	0.018189	0.018189
1000	0.018149	0.018149

Coding Effort

The attached code is 100% all written by me. I did not refer to any source code from GitHub or any site.

Also, the data generation is all generated by me based on python random library, gaussian library, and binomial library. **NO** external tool or third party software is used to generate the task sets.