End to End Dynamic Round Robin (E-EDRR) Scheduling Algorithm Utilizing Shortest Job First Analysis

Renz Rallion T. Gomez
Author/Student
University of the Cordilleras
Baguio City, Philippines
decemberavis19@gmail.com

Christopher M. Bermudez
Author/Student
University of the Cordilleras
Baguio City, Philippines
tupz0799@gmail.com

Vily Kaylle G. Cachero
Author/Student
University of the Cordilleras
Baguio City, Philippines
cacherokaylle@gmail.com

Eugene G. Rabang
Author/Student
University of the Cordilleras
Baguio City, Philippines
raterkracks@gmail.com

Rey Benjamin M. Baquirin
Co-Author/Professor
University of the Cordilleras
Baguio City, Philippines
reybenbaq@gmail.com

Roma Joy D. Fronda
Co-Author/Professor
University of the Cordilleras
Baguio City, Philippines
rdfronda@yahoo.com

Eugene Frank G. Bayani
Co-Author/Professor
University of the Cordilleras
Baguio City, Philippines
ebayani@gbox.adnu.edu.ph

ABSTRACT

The End to End Dynamic Round Robin (E-EDRR) Scheduling Algorithm Utilizing Shortest Job First Analysis functions as a method of queuing and allocating time to tasks that the CPU will process. It is an improved Round Robin that uses Shortest Job First to compare tasks and end to end method to execute tasks. It aims to reduce these three metrics: (1) the time it takes to complete tasks, (2) the time it takes for the ready-for-processing tasks to be executed, and (3) the number of times the CPU switches between tasks. To verify these aims, test cases were conducted that showed the comparative results between E-EDRR and the original algorithms, (Round Robin and Shortest Job First). In the findings, it satisfied the expectations by getting lower scores than both of the original algorithms from all the metrics in all test cases except in one where the Round Robin's variables favor the conditions of the case. It was concluded that E-EDRR has achieved its goal and proved its theoretical acquisitions.

CCS Concepts

• **Theory of computation→Scheduling algorithms**

Keywords

Scheduling Algorithms; Round Robin; Shortest Job First

1. INTRODUCTION

The function of Scheduling Algorithms in Operating Systems is to provide an established method of queuing and allocating time to tasks/instructions for the Central Processing Unit (CPU) to process.

Some basic Scheduling Algorithms are First Come First Serve (FCFS), Shortest-Job-First (SJF), Round Robin (RR), Multilevel Queue Scheduling [1]. These algorithms are globally used in a wide variety of ways. The goal in conceptual capabilities is designing an algorithm that is either faster, more efficient, effective, or a combination of any of the three. These improvements can be achieved by tweaking or combining traits of existing algorithms, or creating a new algorithm. In order to measure these improvements, criteria such as CPU utilization, throughput, turnaround time, waiting time, response time, context switches, etc. are used and compared to other algorithms. In this paper, the focus is to reduce the total turnaround time (TTAT), average turnaround time (ATAT), average waiting time (AWT) and the number of context switches (CS) by making use of the end to end method of processing. In general context, end to end is explained by the relation of the two furthest entities. In this paper, end to end is the method of executing tasks from the first task and jumping to the last one. This can be achieved by combining SJF analysis to the RR method of processing.

Like other scheduling algorithms that use the combination of SJF and RR, the implementation of the original scheduling algorithm is partial and precisely manipulated based on the demands of the goal [5]. The difference in the implementation of End to End Dynamic Round Robin (E-EDRR) Scheduling Algorithm Utilizing Shortest-Job-First Analysis is rather more complex since it queues based on the burst times, executes tasks based on position in the queue, and updates the queues after executions.

To further understand the context of the criteria considered in improving scheduling algorithms, a review of the related literature is appropriate. How E-EDRR is implemented in the simulation is to be explained in the Methods section. E-EDRR is compared to two other existing scheduling algorithms namely SJF and RR since

ICCMB 2020, January 31-February 2, 2020, Tokyo, Japan

ACM ISBN 978-1-4503-7677-8/20/01…$15.00
DOI: https://doi.org/10.1145/3383845.3383869

they are the most related to the proposed algorithm. The test cases that will be used in the simulation will be defined in the Test Cases section. Conclusions and Recommendations will be provided after the analysis of findings in the Discussions section.

2. REVIEW OF RELATED LITERATURE

Most of the improved CPU scheduling algorithms are focused on the improvement of the implementation of the available resources. These resources are always of the consideration of the user and is capable of determining the criteria in measuring the various algorithms' performances. These criteria include:

Turnaround Time: The time required to complete a process (wall clock time). It starts from submission time to completion.

Waiting Time: The time that a process spends in the queue before being executed.

Context Switch: The process of switching tasks/thread, given that the current process is saved so it can be continued later on.

Improved CPU scheduling algorithms are usually in the form of combined existing scheduling algorithms but making some simple to complex changes in the algorithms' structure. Some of the most popular existing algorithms follows:

1. The First Come First Serve (FCFS) or also known as First In First Out (FIFO) uses the queue method of scheduling wherein the first to arrive is the first to leave. In reality, it is represented by a line or a lining system.
2. Shortest Job First (SJF) analyzes the tasks and executes the shortest one first.
3. Round Robin (RR) consecutively executes each task equally given a TQ.
4. Best Job First (BJF) queues tasks based on the tasks Priority, Arrival Time and Burst Time (given factor f where:

f = Priority + Arrival Time + Burst Time ----- [*equation 1*]

to determine tasks location in the ready queue).

These are just a few of the wide variety of scheduling algorithms but given analysis and testing, BJF has succeeded in satisfying a good performance in the scheduling policy [2]. When dealing with all types of job with optimum scheduling criteria, SJF serves but tends to result in increased waiting time for long processes [3, 4]. Combining SJF and RR makes it so that each unique characteristic can be used in the benefit of the hybrid algorithm. For example, using the SJF's analysis on the tasks can determine the shortest burst time and be used as the RR's constant TQ to execute the tasks consecutively [5].

Considering the SJF and RR hybrid algorithm (Enhanced round robin CPU scheduling with burst time based time quantum [5]), the researchers present an improvement of it by using a dynamic TQ based on the current shortest task's burst time and context switching only between the current longest and shortest tasks. The researchers henceforth refer to this as E-EDRR.

3. METHODS

3.1 Assumptions

a. Burst times are known.
b. A batch of tasks or an individual task is received and the algorithm is applied.

3.2 Interpretation of the E-EDRR Algorithm

a. Ready queue (Q1) is the queue that holds the tasks that are ready for execution.
b. Tasks in Q1 are sorted based on their burst time.
c. Time quantum (TQ) is calculated by:

TQ = current shortest task's burst time ----- [*equation 2*]

d. Algorithm is applied on the Q1 and is reapplied until Q1 is empty.
e. Shortest and longest tasks are executed consecutively.
f. If upon execution, the longest task is incomplete, the longest task's progress is saved and the burst time is reduced by TQ.
g. Completed tasks are removed from Q1.
h. Newly arrived tasks are added to the Q1 and updated.

3.3 Pseudocode of the E-EDRR Algorithm

Let TQ be the time quantum.
Let NA be the newly arrived processes.
Let Q1 be the ready queue

1. if(NA == true) {enqueue NA to Q1,
   repeat step 1}
   else {proceed to step 2}
2. if(Q1 != empty) {sort tasks according to burst time,
   proceed to step 3}
   else {proceed to step 1}
3. Determine the TQ by using [*equation 2*]
4. if(Q1.length != 1) {execute shortest task,
   execute longest task}
   else {execute shortest task}
5. if(longest task != complete) {Save longest task's progress
   its burst time is reduced by TQ}
   else {proceed to step 6}
6. Dequeue completed tasks from Q1 and proceed to step 1

3.4 Illustration of the E-EDRR Algorithm

In addition to the 3.2 Interpretation of the E-EDRR Algorithm, the researchers have provided a simulation. The table below shows 5 tasks. The burst times of each process are known and presented.

Table 1: Simulation

Task	Burst Time
T0	16
T1	12
T2	20
T3	26
T4	22

Processes are sorted and added to Q1. Currently, Q1 consists of T1, T0, T2, T4, and T3, necessarily in that order. In relation to the length of tasks, the current shortest task is T1 and the longest is T3. The TQ can now be determined based on T1's burst time which is 12. T1 is executed. T3 is executed with the same TQ. T3's progress is saved and its burst time is now 14 (26 - 12 = 14). Q1 updates. T1 is removed from Q1. Currently, Q1 consists of T3, T0, T2, and T4, necessarily in that order. The current shortest task is T3 and the current longest is T4. TQ is updated and is now T3's burst time which is 14. T3 is executed. T4 is executed with the same TQ. T4's progress is saved and its burst time is now 8 (22 - 14 = 8). Q1 updates. T3 is removed from Q1. Currently, Q1 consists of T4, T0, and T2, necessarily in that order. The current shortest task is T4, and the longest is T2. TQ is updated and is now T4's burst time which is 8. T4 is executed. T2 is executed with the same TQ. T2's progress is saved and its burst time is now 12 (20 - 8 = 12). Since there are no newly arrived tasks, the algorithm now updates the queues. T4 is removed from Q1. Currently, Q1 consists of T2 and T0, necessarily in that order. The current shortest task is T2 and the longest is T0. TQ is updated and is now T2's burst time which is 12. T2 is executed. T0 is executed with the same TQ. T0's progress is saved and its burst time is now 4 (16 - 12 = 4). Q1 updates. T2 is removed from Q1. Currently, Q1 consists of T0 only. Automatically, it is defined as the shortest task. TQ is updated and

is now T0's burst time which is 4. T0 is executed. Q1 updates. T3 is removed from Q1. Since Q1 is now empty, the algorithm will be on standby.

4. TEST CASES

All test cases were performed with the consideration of the following assumptions:

1. Processes are executed in a single processor.
2. Processes are CPU bound.
3. Number of processes and BTs are initially known.
4. RR will have a TQ of 25.

The metrics used for algorithm comparison are the total turnaround time (TTAT), average turnaround time (ATAT), average waiting time (AWT), and number of context switches (CS). TTAT refers to the amount of time all processes take from submission to completion. ATAT is the TTAT divided by the number of processes. AWT is the average amount of time that processes spend in the queue before being executed.

Test Case 1: We assumed five (5) processes wherein they have equal BTs (as shown in Table 2 below).

[Table 2: Test Case 1]

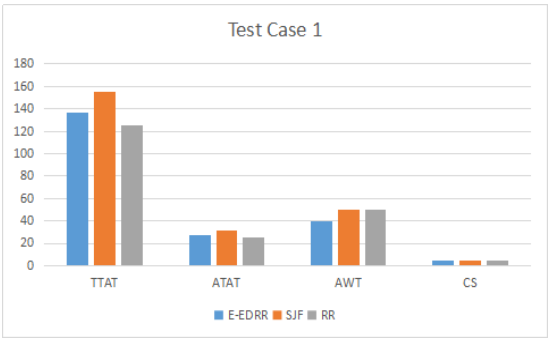
Task	Burst Time
T0	25
T1	25
T2	25
T3	25
T4	25

Table 3 shows the comparative results of E-EDRR against the benchmarking algorithms.

[Table 3: Test Case 1 Results]

Algorithm	TTAT	ATAT	AWT	CS
E-EDRR	137	27.4	40	5
SJF	155	31	50	5
RR	125	25	50	5

E-EDRR scored better than SJF but worse than RR in TTAT and ATAT. In AWT, E-EDRR scored better than both of the algorithms. In CS, the scores are the same across all algorithms. Figure 1 below shows the said results.



[Figure 1: Test Case 1 Results Chart]

Test Case 2: We assumed five (5) processes wherein their BTs in increasing order (as shown in Table 4 below).

[Table 4: Test Case 2]

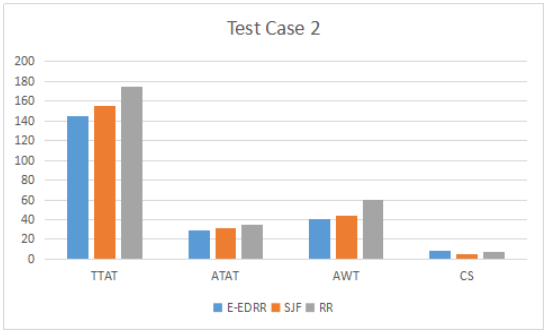
Task	Burst Time
T0	19
T1	22
T2	25
T3	28
T4	31

Table 5 shows the comparative results of E-EDRR against the benchmarking algorithms.

[Table 5: Test Case 2 Results]

Algorithm	TTAT	ATAT	AWT	CS
E-EDRR	145	29	40.6	9
SJF	155	31	44	5
RR	175	35	60	7

E-EDRR scored better than SJF and RR in TTAT, ATAT and AWT. In CS, E-EDRR scored worse than SJF and RR. Figure 2 below shows the said results.



[Figure 2: Test Case 2 Results Chart]

Test Case 3: We assumed five (5) processes wherein their BTs in decreasing order (as shown in Table 6 below).

[Table 6: Test Case 3]

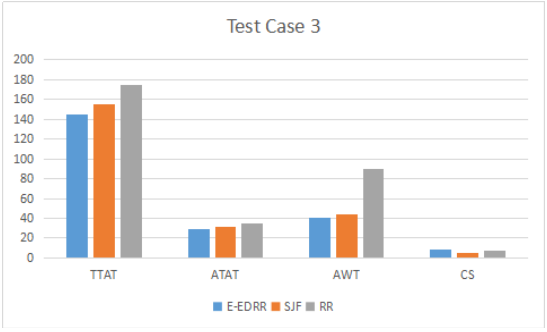
Task	Burst Time
T0	31
T1	28
T2	25
T3	22
T4	19

Table 7 shows the comparative results of E-EDRR against the benchmarking algorithms.

[Table 7: Test Case 3 Results]

Algorithm	TTAT	ATAT	AWT	CS
E-EDRR	145	29	40.6	9
SJF	155	31	44	5
RR	175	35	90	7

E-EDRR scored better than SJF and RR in TTAT and ATAT. In the AWT, RR doubled the score of E-EDRR and SJF. In CS, E-EDRR scored worse than SJF and RR. Figure 3 below shows the said results.



[Figure 3: Test Case 3 Results Chart]

Test Case 4: We assumed five (5) processes wherein their BTs in random order (as shown in Table 8 below).

[Table 8: Test Case 4]

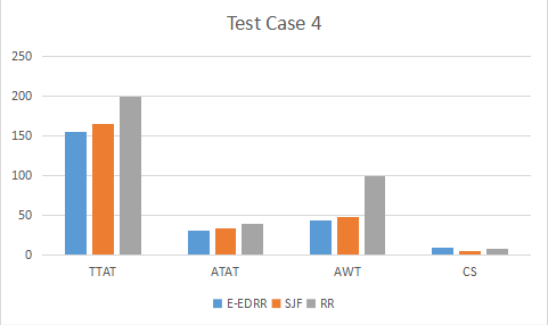
Task	Burst Time
T0	27
T1	21
T2	29
T3	34
T4	24

Table 9 shows the comparative results of E-EDRR against the benchmarking algorithms.

[Table 9: Test Case 4 Results]

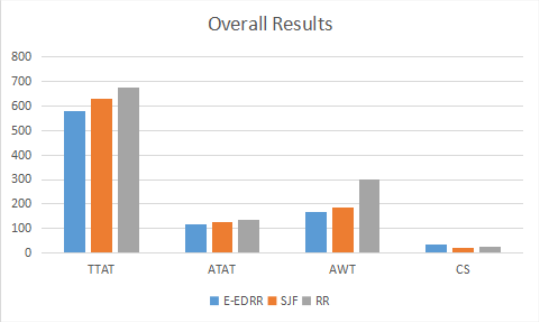
Algorithm	TTAT	ATAT	AWT	CS
E-EDRR	155	31	44.2	9
SJF	165	33	47.8	5
RR	200	40	100	8

E-EDRR scored better than SJF and RR in TTAT and ATAT. In AWT, RR doubled the score of E-EDRR and SJF. In CS, E-EDRR scored worse than SJF and RR. Figure 4 below shows the said results.



[Figure 4: Test Case 4 Results Chart]

In summary of the findings, the E-EDRR scored better in terms of TTAT, ATAT and AWT while being equal to SJF in # of CS but still better than RR. This overall observation is illustrated in Figure 5 below.



[Figure 5: Overall of Test Cases Results]

5. DISCUSSIONS

Based on the results of the test cases, the End to End Dynamic Round Robin (E-EDRR) Scheduling Algorithm Utilizing Shortest Job First Analysis improves total and average turnaround time (TTAT and ATAT), and average waiting time (AWT) but compromises in context switching (CS).

Upon further analysis, the E-EDRR is shown to have better performance in processes with various BTs (as shown in test cases 2, 3, and 4) in TTAT and ATAT. However, test case 1 is the only case where E-EDRR is worse than RR in TTAT and ATAT. This has occured due to that RR was given an advantage in test case 1 because the conditions of test case 1 favored the characteristics of RR. RR's TQ was equal to the BTs of the processes. In overall results, E-EDRR performed better than SJF and RR in TTAT and ATAT by approximately 8% and 16%, respectively. Across all test cases, the E-EDRR performed better than SJF and RR in AWT by approximately 12% and 81%, respectively. Across all test cases, the E-EDRR performed worse than SJF and RR in CS by approximately 6% and 19%, respectively. In addition, E-EDRR had greater number of CS than SJF and RR which is explained by E-EDRR's characteristic of dynamic TQ based on the shortest instruction. Furthermore, E-EDRR, in its essence, is more complex than than both SJF and RR since it is a hybrid of both.

6. CONCLUSIONS

Comparative results from the test cases showed that E-EDRR improves the CPU scheduling by reducing turnaround and waiting time but compromising in context switching. These results are in alignment and validated with the researchers' conceptual analysis of the algorithm before the actual testing was conducted. In respect to the structure of the E-EDRR, the researchers have confirmed that the algorithm's procedural execution of processes is better than the simpler structure of SJF and RR in all types of processes given that the RR's TQ is not acquired based on given test cases. Therefore, the researchers conclude that E-EDRR has achieved its goal in reducing turnaround and waiting time but failed in reducing the context switching.

7. RECOMMENDATIONS

In terms of improvement (i.e. Threading implementation), the E-EDRR can show even better performance. Additionally, for future references, test cases can include the arrival time as another variable in describing tasks. Furthermore, another method (other than end to end) can be implemented and tested for comparison to the original algorithm.

8. REFERENCES

[1] A. Silberschatz, P. B. Galvin and G. Gagne 2004 Operating System Concepts *John Wiley and Sons, Inc. 2005* (Yale University, Corporate Technologies, Inc., Westminster College)

[2] Mohammed Abbas Fadhil Al-Husainy 2007 Best-Job-First CPU Scheduling Algorithm (Middle East University, Amman, Jordan) DOI: 10.3923/itj.2007.289.293 Source: DOAJ *Information Technology Journal-2007*

[3] Pushpraj Singh, Vinod Singh and Anjani Pandey 2014 Analysis and COmparison of CPU Scheduling Algorithms *IJETAE Volume 4, Issue 1, January 2014* (VITS Engineering College Satna (MP), India)

[4] Mario Jean Rene and Dimitri Kagaris 2014 Equitable Shortest Job First: A Preemptive Scheduling Algorithm for Soft Real-Time Systems *OpenSIUC 2014* (Southern Illinois University Carbondale)

[5] J. R. Indusree and B. Prabadevi 2017 Enhanced round robin CPU scheduling with burst time based time quantum *14th ICSET-2017* (VIT University, Vellore, India)