ISSN:0975-9646

Half Life Variable Quantum Time Round Robin (HLVQTRR)

Simon Ashiru , Salleh Abdullahi , Sahalu Junaidu

Department of Mathematics, ABU Zaria,

Nigeria

***Abstract*— Round Robin (RR), one of the oldest CPU scheduling algorithms has found its importance in time sharing systems. In an ideal RR an equal quantum time is allocated to each process residing in the ready queue. A process which is assigned a CPU may run to completion if and only if its quantum time is greater than its CPU burst. Otherwise, after the process exhausted its quantum time the process must be preempted to take turn in the next round. Some major challenges in classical RR are: poor response time, unnecessary context switching and poor multiprogramming. Using Half Life Variable Quantum Time Round Robin (HLVQTRR), variable quantum time is used to eliminate those challenges. All dataset used for the evaluation are generated using normal distribution function.**

***Keywords*— Round Robin (RR), turnaround time, waiting time, context switching (CS), quantum time (QT).**

I. INTRODUCTION

One of the most important functions of an operating system is resource allocation. Since the number of processes ready and requesting to use the CPU is enormous, the operating system must provide a mechanism to carefully distribute resources to processes that are in need. Since the CPU can only attend to a process at a giving time, resources must be shared in some fashion to processes which are ready and requesting to use the CPU. This will provide optimal performance of the system. The technique of sharing this resource is called scheduling and it is performed by a module in the operating system called the scheduler.

There are basically three types of processor scheduler: the job scheduler (***Long term scheduler***), CPU scheduler (***Short term scheduler***) and the ***Medium time scheduler***. The long term scheduler selects jobs from the job pools into memory for execution. It determines which programs are admitted to the systems for processing, thus, it controls the degree of multiprogramming [6]. But the short term scheduler selects processes from the memory and assigns them CPU for execution. Scheduling here is based on the requirement of the resources. It is essentially concern with memory management and often designed as a memory management subsystem of an operating system [5]. Some operating systems such as time sharing systems, may introduce an additional scheduling known as ***Medium time scheduler***. The sole idea of medium term scheduling is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it is left off [10]. It temporarily removes a process from the main memory which is of low priority or has been inactive for a long time. This scheme is called ***swapping.*** The process is swapped out, and is later swapped in, by the medium-term scheduler [10].

Fig. 1 below shows the abstraction on how jobs are admitted into memory by the job scheduler or long term scheduler. Then, the processor scheduler selects one among many processes and execute.

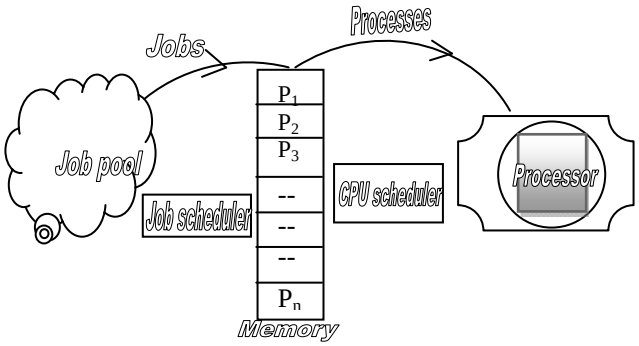


Fig 1: Abstraction of scheduling structure

A. Preliminaries

A process is defined as an active program. A program loaded into main memory for execution is considered to be a process. Programs are passive entity, such as a file containing a list of instructions stored on disk [10]. A program becomes a process when an executable file is loaded into memory for execution [10]. A process is said to be active because besides the code section, it also include the current activities, as represented by the value of the program counter and the contents of the processor's registers [5]. A giving process may be in the following states:

- ***New:*** a process is just newly created.
- ***Running:*** the process is currently using the CPU.
- ***Waiting:*** the process is waiting for some event to occur such as I/O event and so on.
- ***Ready:*** the process is ready to be assign a processor.
- ***Terminated:*** the process has finished execution.

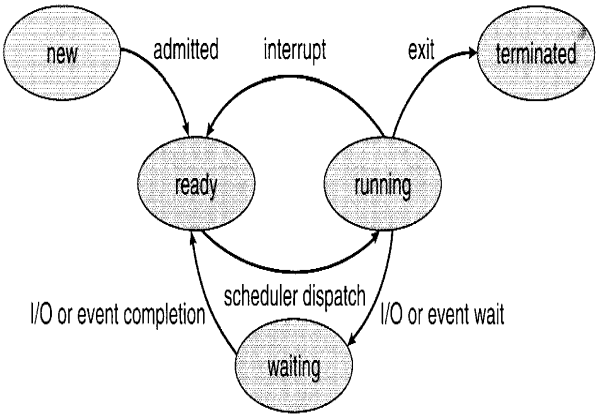


Fig. 2: Process state diagram

Fig. 2 shows that a new created process is admitted into the ready queue. Once admitted, its state changes from 'new' to 'ready' state. As soon as a ready process is dispatched to use the CPU, its state will become 'running'. Three things can happened to a running process: it may be interrupted as a result of the timer interrupt and its state become 'ready'; or a running process can upon completion exit the system and its state becomes 'terminated'; or a running process may request for an i/o event which will be forced to release the CPU and it state becomes 'waiting', and then back to 'ready' upon completion of i/o event.

II. SCHEDULING CRITERIA

Some of the various parameters use to measure or evaluate the performance of CPU scheduling algorithms are listed below.

- CPU utilization: The CPU must not be idle, it should be as much as possible 100% busy.
- Throughput: The amount of tasks/jobs to be completed within a given time is known as throughput. Throughput should be maximized.
- Turnaround Time: it is the time taken to complete a given job. In order word, it is the time a job is waiting in the ready queue plus its CPU burst. Turnaround time is to be minimized.
- Waiting Time: This is the time a job waited in the ready queue. It should be minimized.
- Response Time: In an interactive system, turnaround time may not be the best criterion. Often, a process may produce some output fairly early and continue computing new results while previous results are being output to the user. Response time is the time from the submission of a request to when the first response is produced. This also should be minimized.
- Number of context switching: This is the act of switching a process in and out of the CPU as a result of interrupt. It should be minimized.

III. MOTIVATION

Quantum Time (QT) is the major challenge of RR CPU algorithm because if one is not careful enough the algorithm may change and its purpose is defeated. Choosing a larger QT will practically degrade the system to First Come First Serve (FCFS) scheduling, a smaller one will create an overhead of context switching. Majority of the research work on dynamic RR jerks up QT above the average so as to achieve reduction in the average waiting time, average turnaround time and number of context switching. This is good but it is bound to face even greater challenges such as: poor multiprogramming, poor response time and unnecessary context switching. Some of the proposed algorithms went further and attached priority to shorter jobs. These two key issues make most of the proposed algorithms tending toward FCFS and Shortest Job First (SJF), and at the same time causing unnecessary context switching and poor response time. In the proposed algorithm, variable QT such that each is far less than the average of processes in the ready queue shall be computed. This shall greatly improve on the response time, multiprogramming, and eliminate unnecessary context switching.

IV. RELATED WORKS

Each of the proposed dynamic RR CPU scheduling algorithms provides some level of solutions base on their arguments. Among which are: Even Odd Round Robin (EORR). EORR took note of positions of processes in ready queue. The average of processes in odd positions is compared with the average of processes in the even positions and the greater is considered to be the quantum time [6]. Average Mid Max Round Robin (AMMRR). AMMRR calculates its quantum time as the mean between the average of processes in the ready queue and maximum CPU burst time [8]. In Ascending Quantum Minimum Maximum Round Robin (AQMMRR), quantum time is gotten by multiplying the summation between the minimum and the maximum CPU burst by 80 percent [2]. As for Multi Dynamic Quantum Time Round Robin (MDQTRR), there are two quantum time in each round. The first process up to the middle process uses the median quartile formula to calculate quantum time while preceding processes uses third quartile formula for it quantum time [4]. In Variable Quantum Time (VQT), averaging technique is employed to ensure that each process in the ready queue has a different quantum time [13]. Dynamic Quantum Time using the Mean Average compute time quantum in each round and use it to perform scheduling to processes in ready queue [1].

V. PROPOSED APPROACH

Half Life Variable Quantum Time RR (HLVQTRR): Sometimes, some processes may go for more than two round and as such increases the number of context switching, average waiting time and average turnaround time. But HLVQTRR ensures that half of every processes' bursts be executed in the first round, and in the second round the remaining half should run to completion. This will greatly improve multiprogramming irrespective of the variation in the processes' length in the ready queue. In SJF, the major problem is starvation. This is solved because only half of a process's job is executed what is left is preempted and the next process is attended to and so on. The major

emphasis of this scheduling is: high multiprogramming, high response time, simplicity, elimination of unnecessary context switching and fairness which are the main criteria of RR CPU scheduling algorithm. At least, whatever the case may be, response time for each process will be high. This attribute is of high interest to real time systems. Another advantage of HLVQRR is that it maximizes the cost of calculating quantum time in the second round. This means that in the second round which is also the final round, all the processes in the first round reappear. Other algorithms may go for second and third round with just two or three processes out of many. This may cause a serious disadvantage. Fig. 3 and Fig. 4 below are the flow chart and algorithm for the proposed approach

.

Fig. 3 : HLVQTRR flow chart

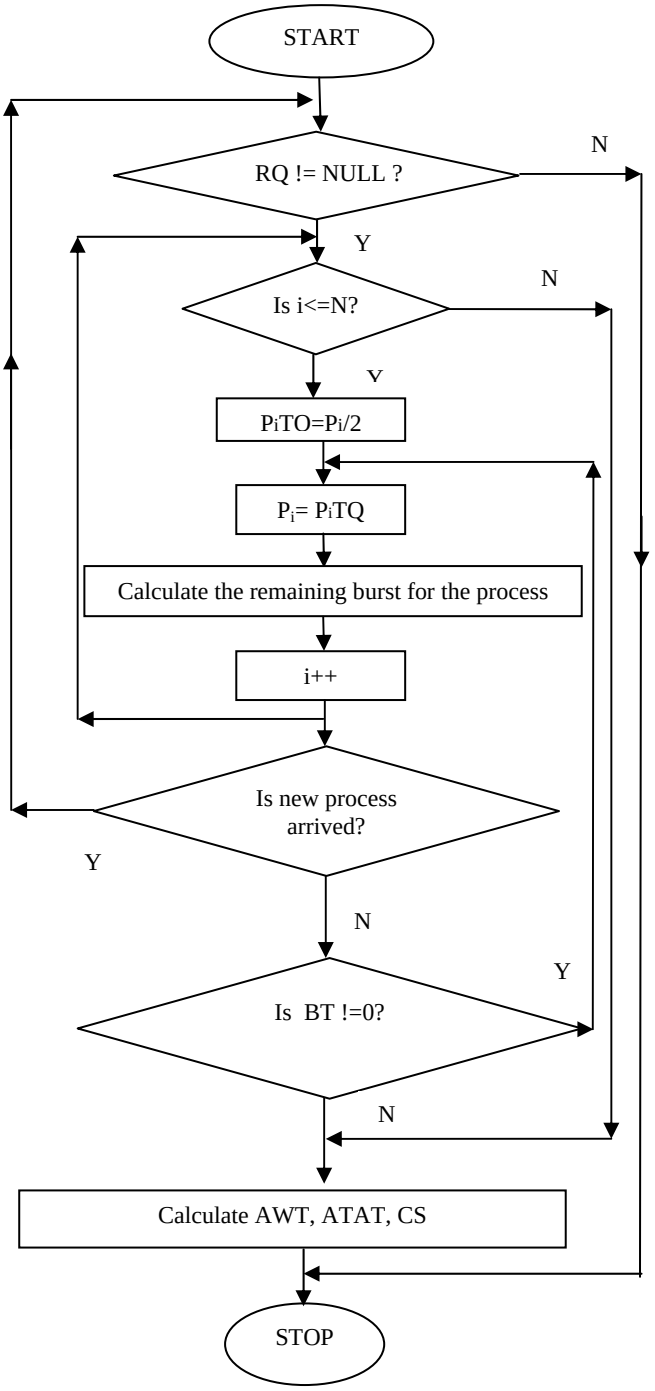


Fig 4: HLVQTRR algorithm

1. //N= Number of processes

//Pi= i^{th} Process

//P_iQT= Quantum time for i^{th} process

//i= Loop variable

//QT = quantum time

//BT= Burst time of the processes

2. While(RQ !=NULL)

// RQ= Ready Queue

//Calculation of Quantum time (QT)
for i=1 to N Loop
{
P_iQT= P_i/2 // take the floor value
}

3. // Assign QT to (1 to n) processes.
for i=1 to N loop
{
P_i=QT
Calculate the remaining Burst time of i^{th} process.
}
End of for

4. If (new process arrived)
Then go to step1
else if (new process is not arrived and BT!=0)
then go to step 3
else
goto step 5
end of if
end of while

5. Calculate AWT, ATAT and CS
//AWT=Average waiting time
//AWT=Average turnaround time
//CS= Number of context switch

6. End

VI. ILLUSTRATION/ANALYSIS

Case 1:

Using mean =80 and deviation=20, the following processes *(P_i)* and their associated CPU burst are generated. {P1=88, P2=89, P3=85, P4=93, P5=90, P6=84, P7=90}

A. Classical Round Robin

In a classical RR, the quantum time is the average of processes CPU burst in the ready queue.
Quantum time (QT) = (88+89+85+93+90+84+90)/7 =619/7 = 88. (Always use the ceiling.)
After applying Round Robin, the left over time will be:
Remaining processes CPU burst: P2=1, P4=5, P5=2, P7=2. These lefts over will be use in round two (2) with same QT.

1) Observation: ***In classical RR,***

Multiprogramming: Only four (4) processes display multiprogramming because they reappeared in the next round. They are: P2, P4, P5 and P7, while the rest ran to completion.

Response time: For P2, after the first round it began to respond at time 609, P4 at time 610, P5 at time 615 and P7 at time 617. This is not good at all.

Context switching: Context switching is 11. The overhead cost incurred performing context switching in the second round was unnecessary. P2 was preempted just for 1 time unit, P4 was preempted just for 5 time unit, P5 was preempted just for 2 time unit and P7 was preempted just for 2 time unit. All of these were unnecessary. If it can execute up to 88 time unit, it should not have been preempted just for 1 remaining time unit. Same goes to the rest. It also implies that the multiprogramming it displayed was also unnecessary.

B. Half Life Variable Quantum Time Round Robin (HLVQTRR)

Method: (take the ceiling to be QT)
P1: QT= 88/2 = 44, P2: QT= 89/2≈ 45**,** P3: QT= 85/2 ≈ 43**,** P4: QT = 93/2≈ 47**,** P5: QT= 90/2 = 45**,** P6: QT= 84/2 = 42**,** P7: QT = 90/2 = 45. After applying round robin, the left over time will be:
Remaining processes CPU burst: P1=44, P2=44, P3=42, P4=46, P5=45, P6=42, P7=45. In this second round, each process will run to completion. That is QT will be equal to each processes' remaining CPU burst.

1) Observation: ***In HLVQTRR,***

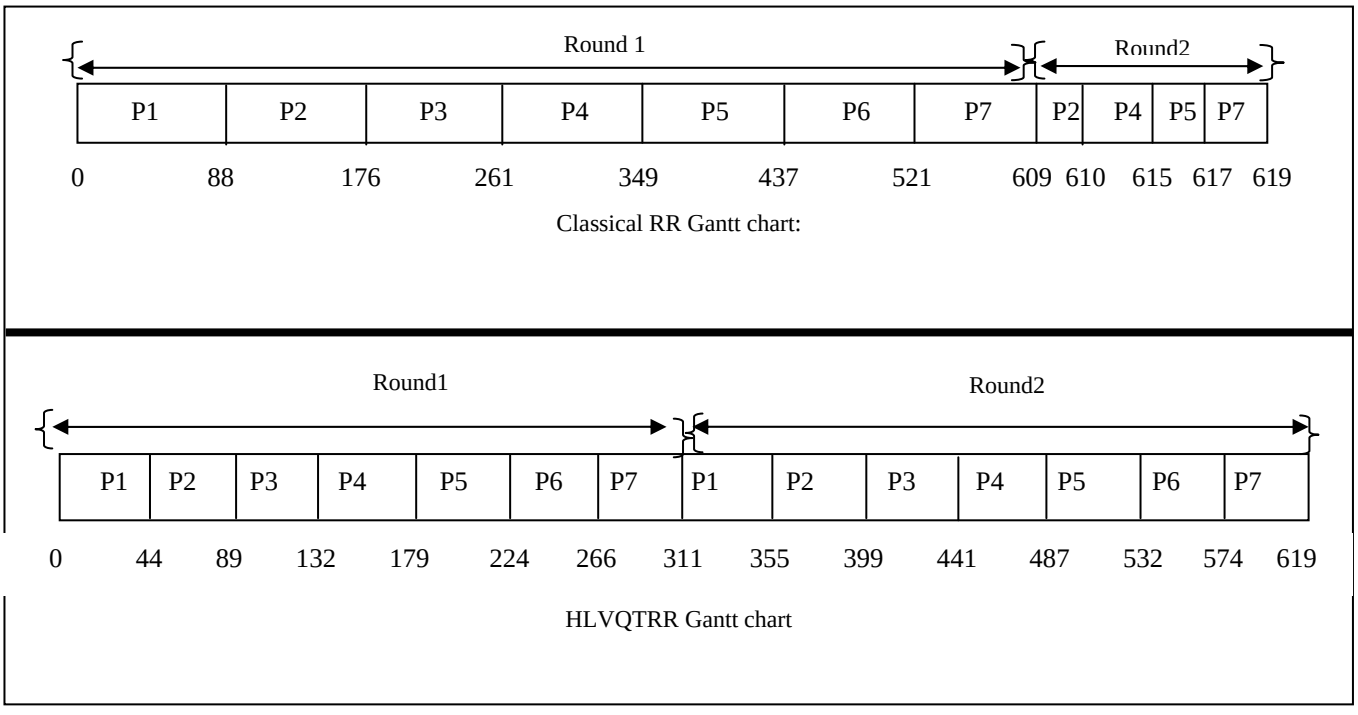
Multiprogramming: All the processes will always display multiprogramming. Multiprogramming is guaranteed.

Response time: P2 began responding at 355, P4 at 441, P5 at 487 and P7 at 574 time unit. Compare to the classical RR, the difference in the response time for P2 between classical RR and HLVQT RR is 609-355=254 time unit. Their differences are highly significant in favour of HLVQTRR. This goes same with the rest processes.

Context switching: Context switching is 14. Although it is much, it has achieved high response time. Each of the switching is done intelligently because processes must have executed half way before switching.

Fig. 5 below shows the Gantt charts between Classical RR and HLVQTRR.

Fig 5: Gantt chart of classical RR and HLVQTRR



Using mean =80 and deviation=60, the following processes and their associated CPU burst are generated. P1=110, P2=89, P3=113, P4=137, P5=86, P6=131, P7=95}

Multiprogramming: In Classical RR, P1, P3, P4 and P6 displayed multiprogramming. While In HLVQTRR, all the processes displayed multiprogramming.

Response time: In classical RR, P1 start responding at 706, P3 at 707, P4 at 711, P6 at 739. While in HLVQTRR, response times are at: P1 at 383, P3 at 482, P4 at 538 and P6 at 649

Context switching: In classical RR, their lefts over time were so small. Switching for next round was unnecessary. While in

HLVQTRR, Context switching is 14. Each of the switching was done intelligently.

Case 3:

Using mean =80 and deviation=70, the following processes and their associated CPU burst are generated. **{**P1=82, P2=128, P3=113 P4=129, P5=115, P6=115, P7=81}

Multiprogramming: In Classical RR, P2, P3, P4, P5 and P6 displayed multiprogramming. While In HLVQTRR, all the processes displayed multiprogramming.

Response time: In classical RR, P2 starts responding at 708, P3 at 727, P4 at 731, P5 at 751 and P6 at 757. While in HLVQTRR, response times are at: P2 at 425, P3 at 489, P4 at 545, P5 at 609 and P6 at 649

Context switching: In classical RR, their lefts over time were so small. Switching for next round was unnecessary. While in HLVQTRR, Context switching is 14. Each of the switching was done intelligently.

VII. SIMULATION

All processes are considered to be in the ready queue and their arrival time set to be zero. Also, processes are considered to be of same priority. The simulator uses normal distribution function which requires three input parameters to generate processes and their associated CPU burst. The input parameters are: mean (μ), standard deviation (σ), and number of process. When the program is executing, it will request the user to supplier number of process, the mean and the standard deviation. The simulator computes the average waiting time, average turnaround time, and number of context switching for each algorithm (Classical RR and HLVQTRR). The results of the experiments are given in the table and figure below. Fig 6 below shows the interface of the simulator

.

Fig 6: Interface of the simulator

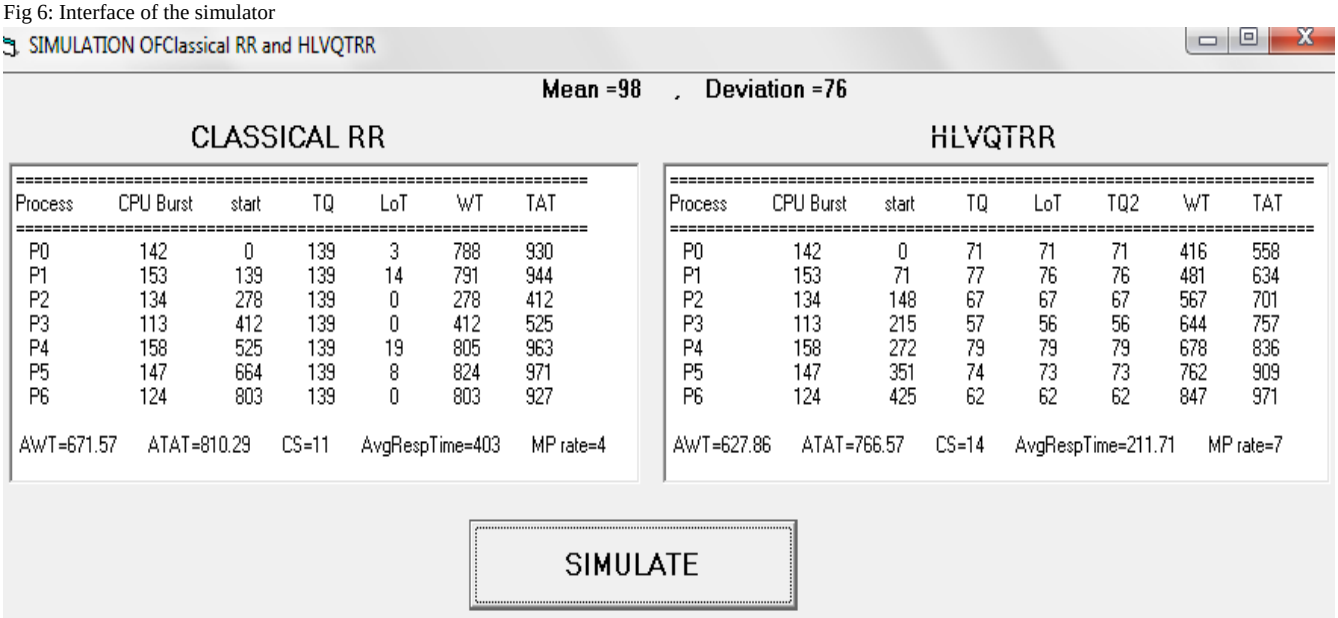


Table 1 below shows the simulation result in a tabular form. *P* is number of process, *d* is the deviation value used in each run, *CS* is the context switching, *MP* stands for multiprogramming rate. The average response time used is the average of the time of response of all the processes in the first round. Multiprogramming tracks those processes that went beyond the first round. From the table, the rate at which HLVQTRR is winning in multiprogramming and response time is more than it is losing in average waiting time and average turnaround time when compared with the classical RR. The idea is to have an algorithm which has a better response time and multiprogramming such that it does not suffer much in average waiting time and average turnaround time.

Table 1: Simulation results

Simulation result between Classical RR and HLVQTRR										
			Classical RR				HLVQTRR			
P	d	Mean	AWT	ATAT	Average response time	MP	AWT	ATAT	Average Response time	MP
10	45	56	565.4000	648.8000	358.8000	6	564.60000	648.00000	189.3000	10
20	78	98	1724.700	1863.700	1203.300	10	1961.5500	2100.5500	640.3500	20
30	321	213	6690.070	7034.470	4390.970	14	7458.5300	7802.9300	2464.930	30
40	342	453	16518.50	17149.72	11570.18	17	18401.600	19032.800	6092.520	40
50	443	564	29119.96	29941.60	18968.70	25	30284.160	31105.800	10154.42	50
60	543	653	39993.70	40949.23	26142.55	30	42292.900	43248.430	14104.08	60
70	564	674	48017.04	48974.86	30894.93	35	49755.390	50713.200	16710.27	70
80	634	754	59437.34	60493.22	38822.68	38	62810.260	63866.150	21103.35	80
90	746	876	82855.50	84121.70	51997.59	50	84399.130	85665.330	28054.71	90
100	768	987	92664.20	94018.38	61660.62	49	100305.79	101659.97	33276.13	100

The graphs below further demonstrate how efficient HLVQTRR as far as response time and multiprogramming is concern.

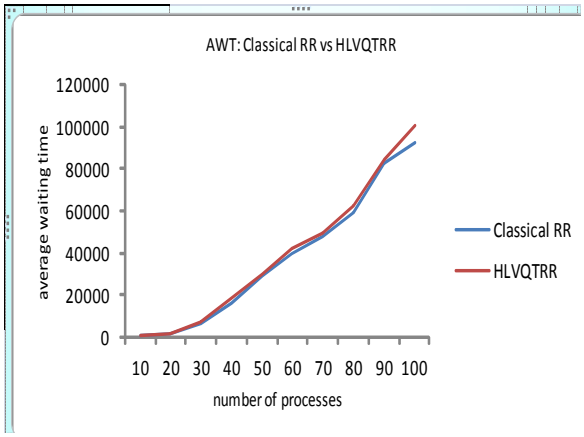


Fig 7: Graph of average waiting time

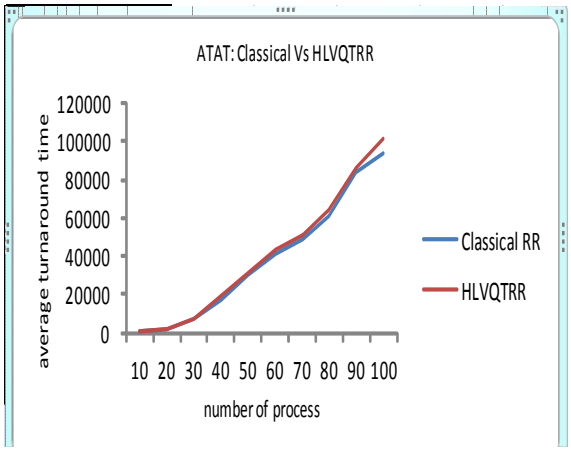


Fig 8: Graph of average turnaround time

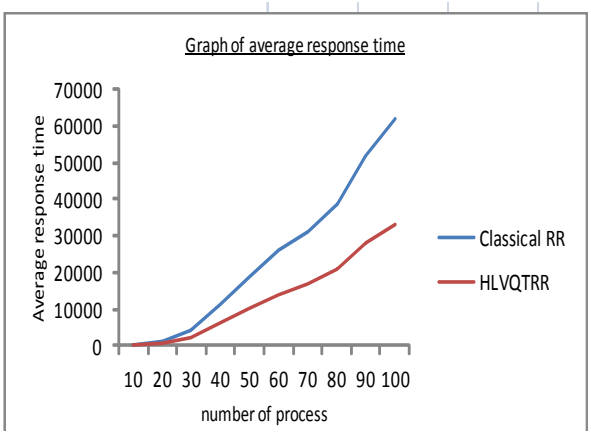


Fig 9: Graph of average response time

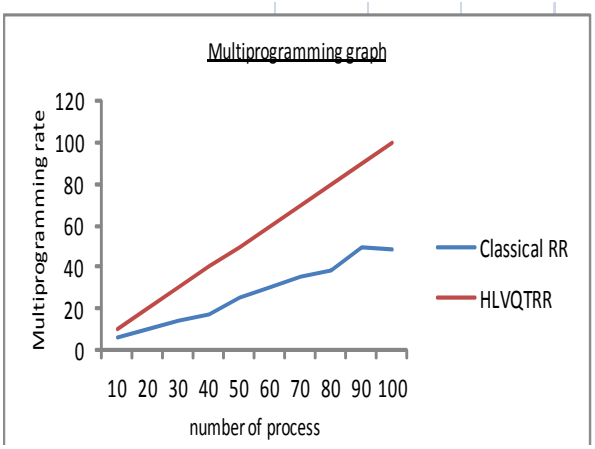


Fig 10: Graph of multiprogramming rate

From fig. 7 and 8 above, Classical RR is slightly better than HLVQTRR. But looking at fig. 9 and 10, it can be observed that HLVQTRR provides a better response time and multiprogramming than its classical RR counterpart. The results obtained in fig. 9 and 10 are so significant unlike the difference obtained in fig. 7 and 8 which is less significant.

VIII. WHY HLVQTRR IS BETTER THAN CLASSICAL RR

In fact, after studying high performance computing, it is clear that HLVQTRR will be a good algorithm to perform scheduling seeing that processes are roughly equally partition. Whatever the case may be, it will demonstrate multiprogramming by ensuring only half of each process is executed in the first round the remaining will be executed in the second round. The slight achievement made by classical RR in average waiting time and average turnaround time against HLVQTRR cannot be compared with the high achievement made by HLVQTRR in response time and multiprogramming. The overall result is in favour of HLVQTRR.

One of the challenging issues regarding classical RR is how it handles multiprogramming in a situation whereby processes with the same priority arrive at different time. For example, if the first process begins using CPU before any other process arrives, irrespective of the process size, the process that has the CPU must run to completion without being preempted. This is because in an ideal classical RR the quantum time will be equal to the CPU burst of the process that arrived first and begins to use the CPU. At this time, the running process cannot be preempted unless another process with higher priority arrives. For example, assuming P1 having CPU burst of 80ms arrives at time t=0, and begin to use the CPU. After 3ms, another process say P2 arrives with CPU burst of 5ms, having same priority with P1. In classical RR, P1 QT will be the same as its burst time, that is QT=80ms. Though it is RR algorithm, in this kind of scenario, the first process which is P1 must run to completion irrespective of its size. P2 will have to wait for as long as it takes. This is a complete drawback in multiprogramming and response time. P2 will begin responding at 80ms-3ms=78ms, a time that will take P1 to complete its execution. This problem is solved in HLVQTRR. The solution is this: if the second process (P2) arrives in the first round of the first process (P1), then P1 will be preempted as soon as it finished its first round. But if P2 arrives in the second round of P1, then the P2 must be patient to wait because the first process (P1) has gone beyond half of its job. Using the same example above, P1 will only run for 40ms, that is, half of its job will be executed in the first round. But, since P2 arrived when P1 must have executed its job for 3ms, P2 will actually begin responding at 40ms-3ms=37ms as compared to 78ms in classical RR. The difference in their response time is 78ms - 37ms=41ms in favour of HLVQTRR.

Another problem with classical RR is that all the processes less than or equal to their average value will definitely run to completion without being preempted. This does not display multiprogramming and response time is very poor. In order to demonstrate high level of multiprogramming and response time, HLVQTRR is introduced. In HLVQTRR, half of each process must return for second round. For example, looking at their Gantt charts (fig. 5) for the classical RR, after first round, P2 began responding at 609, P4 at 610, P5 at 615 and P7 at 617. But when compared with that of HLVQTRR Gantt chart P2 began responding at 355, P3 at 399, P4 at 441, P5 at 487, P6 at 532 and P7 at 614. It is very clear that the difference in their corresponding response time is very high in favour of HLVQTRR.

The other problem is tie up to the attempt made on the various proposed dynamic RRs. It is clear that each of the approach tries to jerk up the quantum time just above the average. Because the classical RR uses the average as its quantum time, all the dynamic RR jerks up the quantum time slightly above the average. This is an attempt to capture more processes in the first round that were not captured in the first round when using classical RR. Their concept is the same: for one dynamic RR to perform better than the other, its quantum time must just be slightly greater. If the quantum time is continuously been jerked up a time will reach when the algorithm will become FCFS. But HLVQTRR provides quantum time for each process that is far lower than the average, and despite that, it demonstrates high rate of multiprogramming involving all the processes and has a fair result in its AWT and ATAT.

IX. CONCLUSION

It is clear that HLVQTRR is better than the classical RR. It has displayed high rate of multiprogramming and response time. There is no unnecessary context switching like that of classical RR. Although, emphasis was not on average waiting time and average turnaround time, yet it produced a fair result. Given any ideal dataset, HLVQTRR will always perform better in terms of: multiprogramming, response time and appropriate context switching.

REFERENCES

[1] Abbas N, Ali K and Seifedine K., 2011. "A New Round Robin Based Scheduling Algorithm for operating systems: Dynamic Quantum using the mean average". *IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No.1. pp. 224-229.*

[2] Ali D.J. 2012. "Improving efficiency of Round Robin scheduling using Ascending Quantum and Minimum-Maximum burst time", *J. of university of anbar for pure science. Vol.6:NO.2. pp.23-27.*

[3] Bashir A., Doja1 M.N., Biswas R., and Alam .M. 2011. "Fuzzy Priority CPU Scheduling Algorithm". *IJCSI International Journal of Computer Science Issues, Vol. 8. pp. 386-390*

[4] Behera H.S., Rakesh M., Sabyasachi S. and Sourav B.K. 2011. "Comparative performance analysis of Multi-dynamic Quantum time Round Robin (MDQTRR) Algorithm with Arrival Time". *Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2. pp. 262-271.*

[5] Bovet D. P, Cesati M. 2006. 3rd edition, "Understanding the Linux Kerne"l, O'Reilly Media, Inc., USA. pp. 1- 923.

[6] Pallab B., Proba B. and Shweta D.S. 2012a. "Comparative Performance Analysis of Even Odd Round Robin Scheduling Algorithm (EORR) using Dynamic Quantum time with Round Robin Scheduling Algorithm using static Quantum time". *International Journal of Advanced Research in Computer Science and Software Engineering. Volume 2. pp. 62-70.*

[7] Pallab B. Proba B. and Shweta D.S. 2012b. "Comparative Performance Analysis of Average Max Round Robin Scheduling Algorithm (AMRR) using Dynamic Quantum time with Round Robin Scheduling Algorithm using static Quantum time", *International Journal of Innovative Technology and Exploring Engineering (IJITEE). Volume-1. pp. 56-62.*

[8] Pallab B., Proba B. and Shweta D.S. 2012c. "Performance Evaluation of a New Proposed Average Mid Max Round Robin (AMMRR) Scheduling Algorithm with Round Robin Scheduling Algorithm". *International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2. pp.143-151.*

[9] Puneet V.K., Nadeem A. and Faridul S.H. 2012. "Efficient CPU Scheduling Algorithm Using Fuzzy Logic". *International Conference on Computer Technology and Science, IPCSIT vol. 47. pp. 13-18.*

[10] Silberschatz A, Galvin P.B. and Gagne .G, 2005, "Operating Systems Concepts", (7th ed), John Wiley and Sons, USA. pp. 1-885.

[11] Sunita B., Bhavik K. and Chittaranjan H. 2011. "Dynamic Task-Scheduling in Grid Computing using Prioritized Round Robin Algorithm". *IJCSI International Journal of Computer Science Issues, Vol. 8. pp. 472-477.*

[12] Tanenbaun A.S. 2008. "Modern Operating Systems". (3rd ed), Prentice Hall, pp. 1104.

[13] Yashasvini S. 2013. "Determining the Variable Quantum Time (VQT) In Round Robin and importance over Average Quantum Time Method", *International Journal of Science, Engineering and Technology Research (IJSETR) Volume 2. pp. 613-617.*