Analytic Approach of Improve Multilevel Feedback Queue Scheduling

Keanu Go
Computer Science University of the Philippines Visayas
Tacloban College
Tacloban City, Philippines
Email: kggo1@up.edu.ph

Babes Ivy Ngoho
Computer Science University of the Philippines Visayas
Tacloban College
Tacloban City, Philippines
Email: bongoho@up.edu.ph

Abstract – **Multilevel feedback scheduling is a kind of process scheduling algorithm where process with available remaining CPU bursts can be moved or shifted between queues to complete its job. In a multilevel feedback queue, its architecture is divided into multiple queues with a priority level. Shifting between queues happens with the corresponding CPU bursts of the process, if the process takes too much CPU time required in that queue it will be shifted to a lower priority queue. The paper aims to configure to improve the current multilevel feedback queue scheduling where problems in the given standard algorithm will have a corresponding solution. Comparison of the state of the processes running in MLFQ scheduling will be analyzing its performance of the configured CPU scheduling technique to make it much better and applicable in the process scheduling in operating systems. Also, this paper describes the interactive processes.**

Keywords— **CPU burst time, feedback analysis, starvation, response time, turnaround time, waiting time, aging, starvation, parameters**

I. Introduction

Currently, many CPU scheduling algorithms are existing like FCFS, SJF, SRTF, Priority Scheduling, Round Robin Scheduling, MLQ and MLFQ. The MLFQ is one of the most well-known approaches to feedback scheduling. The fundamental problem that the MLFQ tries to address is optimizing turnaround time, which can be done by running shorter jobs first, but the OS doesn't know how long a job will run for. Additionally, MLFQ wants to make a system feel responsive to users, therefore minimizing response time as well. While algorithms like Round-Robin are great for response time, they have terrible turnaround time. Similarly, SJF minimizing turnaround time, but has poor response time [8].

Given that we won't know how long a process will run, how can the scheduler learn as the system runs, and make the best scheduling decisions [8]?

The MLFQ is a great example of a system that learns from the past to predict the future. Future predicting approaches are common in CS, and it is critical that they be accurate [8].

The MLFQ has components like other CPU scheduling algorithms. Scheduler is the component of the kernel that selects which process to run next [1]. The scheduler (or process scheduler, as it is sometimes called) can be viewed as the code that divides the finite resource of processor time between the runnable processes on a system. In MLFQ [2][3], processes are scheduled according to their remaining CPU burst time and they are shifted down from queue to queue as they have some remaining CPU burst time [4]. Every queue has unique time slice that gradually increases from upper level queue to lower level queue.

This paper will indicate the necessary rules or actions to consider in some conditions. It will aim to improve the processes involved in the MLFQ scheduling to which the addressing the problems that may encounter may be resolved with certain configuration. Such that the processes running in the operating system may lead to better performance and prevent some issues.

A good scheduling algorithm is the one that can optimize the performance measures. The optimization performance measures are; maximizing CPU utilization, maximizing throughput, minimizing turnaround time, minimizing waiting time, minimizing response time, maximizing scheduler efficiency, and minimizing the context switching.

II. Related Literature

There are ways to improve the performance of the processes entering to the system that lead to many approaches. In paper [2], Reducing MLFQ scheduling starvation with feedback and exponential averaging, the approach that is related to what will aim in this paper is that it shows a technique that avoids MLFQ starvation, which will result to low overhead and will not lead to risks and failure on the scheduling action of interactive and high-priority processes. Adding a second level of feedback to redirect a "safe" amount of CPU time to the lowest-priority queue to prevent starvation of processes in that queue.

Another approach is in paper [6], recurrent neural network has been utilized to find both the number of queues and the optimized quantum of each queue. Also, in order to prevent any probable faults in processes' response time computation, a new fault tolerant approach has been presented. The experimental results show that using the IMLFQ algorithm results in better response and waiting time in comparison with other scheduling algorithms. Here this proposed neural network takes inputs of quantum of queues and average response time and getting the required inputs it takes the responsibility of finding relation between the specified quantum changes with average response time. It can find the quantum of a specified queue with the help of optimized quantum of lower queues. Thus, this network fixed changes and specify new quantum which overall optimize the scheduling time. In the paper [6] smoothed competitive analysis is applied to multilevel feedback algorithm. Smoothed analysis is basically mixture of average case and worst-case analysis to explain the success of algorithms. This paper analyses the performance of multilevel feedback scheduling in terms of time complexity. Any performance enhancing approach can use this approach for performance analysis in terms of time complexity. Another approach in paper [7], multilevel feedback queue scheduling algorithm is implemented in Linux 2.6 kernel and new Linux2.6 scheduler performance compared with the proposed approach. It describes two algorithms elaborately and then for different load of job, which are running in background, this scheduler is applied for calculating the average response time. And to maintain simplicity inverse relationship is maintained between priority of processes and time slice length.

This paper is a real guideline for designer of cognitive scheduling systems. Now to analyze the performance enhancement, the proposed approach is implemented and simulated in Condor [8] which provides high throughput computing environment, that handles job queue mechanism, scheduling policy, priority scheme, resource monitoring and resource management, based upon the policy fixed for execution Condor executes the job when user submits the job.

Another paper [10], efficient implementation of Multilevel Feedback Queue Scheduling, dynamic time quantum is also used which further improves the efficiency of the scheduling. Here the dynamic time quantum of the queues is calculated based on the burst time of the processes. Time quantum of the second queue is the burst time of the $(2n/3)^{th}$ process (where n is the number of processes remaining after the execution in the first queue) and time quantum of the third queue is burst time of the largest remaining process. Thus 66% of the processes get executed in the second queue and remaining processes in the last queue thus preventing the problem of starvation of huge burst time processes.

And in paper [11], involves the design and development of new CPU scheduling algorithm (the Hybrid Scheduling Algorithm using genetic approach). Like any genetic algorithms, it comes with an approach that will find the best variable to which it may result to better outcome which in this paper the variable will be using the fitness function. The idea is to choose an initial random population, calculate the entity fitness using fitness function (based on minimum time), first-rate pairs of best-ranking entity using fitness function to reproduce, breed new generation through crossover and inversion while terminating condition becomes true. Using genetic operators' crossover, fitness function and inversion we get final population.

III. LITERATURE SURVEY

A. Scheduling Algorithms

There are different scheduling algorithms existing today:

a. First-Come, First-Served

First-Come, First-Served (FCFS) [4] is the most basic scheduling algorithm. It simply allocates the CPU to the process that requested it first. All other processes that request the CPU are placed in the ready queue and are served in the order in which they arrive. This algorithm is non-preemptive; once a process is given control of the CPU it will keep it until the process releases it. A process will release the CPU when it has either finished executing, or if it needs to wait for an I/O event to occur before it can resume its execution.

b. Shortest-Job-First

Shortest-Job-First (SJF) [4] examines the amount of time each process is expected to take during its next CPU burst. It uses this information to form a queue, and then gives control of the CPU to the process that is expected to have to shortest next CPU burst. In theory, this algorithm is very efficient, because it minimizes the average waiting time for each process. However, since the next CPU burst time can be very difficult to accurately determine, this algorithm does not always produce the intended results

c. Shortest-Remaining-Time-First

Shortest-Remaining-Time-First (SRTF) [4] is a preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst.

d. Priority scheduling

Priority scheduling [4] algorithms assign each process a priority level, which is represented by a number. Some systems consider low numbers to have high priorities and others designate the high numbers to indicate a higher priority. These priorities can be assigned internally (by the operating system) or externally (by the user). Considering some sort of measurable quantity, such as how many resources each process requires, usually sets internal priority levels. External priority levels reflect a user-specified order of importance for each process. The process with the highest priority is initially given control of the CPU. If the algorithm is preemptive, then the currently running process will be preempted once a process in the ready queue has a higher priority. However, if the algorithm is non-preemptive, then once the process begins running all other processes will have to wait until it releases the CPU, regardless of what their priority levels may be. The major concern with priority scheduling is the concept of starvation. This occurs when a process has such a low priority that it is never given control of the CPU. A solution to this is aging, which gradually increases the priority of a process over time.

e. Round-Robin scheduling

The Round-Robin scheduling [4] algorithm (RR) defines a time quantum, which is the amount of time each process in its queue is allocated the CPU. This algorithm is preemptive and fills its queue using the FCFS method. After a process has exceeded its time limit, it is moved to the end of the queue and the next process in the queue is given control of the CPU. Setting a correct time quantum is critical to the performance of the RR algorithm. If the time quantum is too small, then few processes will complete the first time they are run and much time will be spent performing context switches. Alternatively, if the time quantum is too large, then most processes will finish and the CPU will sit idle for the remainder of the quantum. In both of these cases, the CPU is not being used efficiently. Generally, it is desirable for around 80 percent of the processes to complete before the time quantum is reached.

f. Multilevel queue scheduling

A multilevel queue [4] scheduling algorithm separates the ready queue into a number of different queues. Each of these queues uses their own scheduling algorithm. For instance, there may be separate queues created for CPU-bound and I/O- bound processes. The multilevel queue algorithm is responsible for allocating the CPU to each of these different queues. Any scheduling algorithm can be used to achieve this goal; however, fixed-priority preemptive scheduling is commonly used. If the CPU- bound queue were assigned the higher priority, then the CPU would not be given to the I/O-bound queue until all of the processes in the CPU bound queue had completed. A multilevel queue is useful when the processes can be easily grouped by importance.

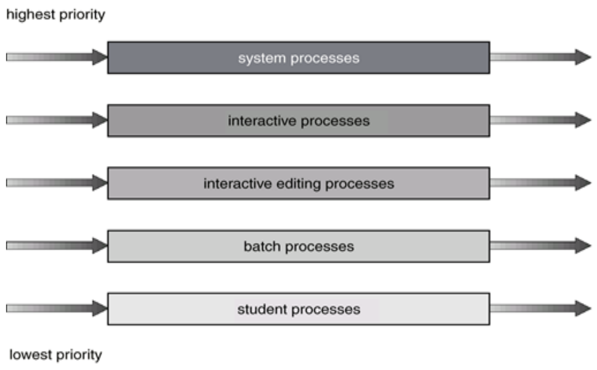

Figure 1: Multilevel queue scheduling

g. Multilevel feedback queue scheduling

In a multilevel queue [4] scheduling processes are permanently assigned to a queue on entry the system. Processes do not move between queues. This setup has the advantages of low scheduling overhead, but the disadvantages of being inflexible. Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. This scheme leaves I/O bound and interactive processes in the higher-priority queues. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher- priority queue. This form of aging prevents starvation.

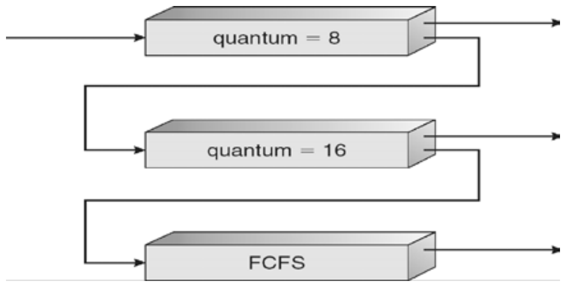

Figure 2: Multilevel feedback queues

A multilevel-feedback-queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithms for each queue.
- The method used to determine when to promote a process.
- The method used to determine when to demote a process.
- The method used to determine which queue a process will enter when that process needs service.

B. Interactive Processes

There are two types of processes, interactive processes and non-interactive processes. The process type is judged according to the cumulative CPU time. If the cumulative CPU time used exceeds the threshold value, then the process is classified as a non-interactive process, otherwise the process is classified as an interactive processes.

Interactive processes are the processes that frequently have long periods of idle time and have relatively little cumulative CPU time that was use between the periods of sleep. CPU time is allocated to the interactive processes with high priority than the non-interactive processes by assigning a high level priority to the interactive type processes [13].

IV. PROBLEM STATEMENT

MLFQ like other scheduling algorithms, problems arise to which processes that are scheduled cause some issues. Starvation is one of the problems that may encounter if some processes are is at a low priority queue and takes up too long to run its process that it will never receive CPU time. Some programs change behavior over time that it is difficult for any scheduling algorithm to predict the future behavior of the program.

It is assumed that if low priority processes are boosted and moved to higher level queue for some time then this will prevent the issue of starvation.

V. PROPOSED SOLUTION

The proposed solution is that there will be actions to avoid problems like starvation. Aging is a technique to avoid starvation by just gradually increase the priority of a process that waits too long. There will be two remedial action that may take to consider its priority. One is that if a process takes too long to wait in a low priority queue it may be moved to the higher priority queue or to the topmost queue. Second, it can also be that the CPU scheduler allocates CPU time for each queue so that processes in the lower level queue can be noticed that it is not purely-based on priority that there will be a chance that a process can be run in given amount of time in a queue. Each queue has a corresponding time slice to which the scheduler acts like a round robin scheduling algorithm bit can be of different time allocation on each queue.

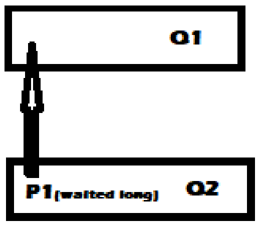

Figure 3: First Approach

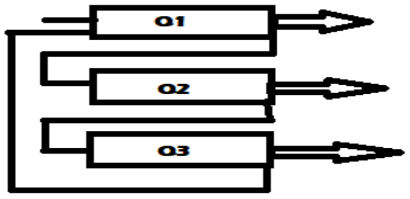

Figure 4: Second Approach

VI. CONFIGURATION

This the configuration of MLFQ with the following parameters to take place [12]:

1. ***Number of queues*** - the simulation shall allow for an arbitrary number of queues provided that it falls within the allowable numbers [1 to max queue].
2. ***Scheduling algorithm for each queue*** - each scheduling queue is associated with its own scheduling algorithm that determines the process to run among a number of process in ready state.
3. ***Migration policy*** - the migration policy determines how a process moves from one queue to another queue and can be classified as:
   a. ***Promotion*** - promotion happens when a process is migrated from a lower priority to a higher priority queue. Default: A process is promoted if it is preempted before it consumes its allocation. Note that if the associated scheduling policy is non-preemptive, the process may still be preempted if the overall queue priority policy is through fixed time slots.
   b. ***Demotion*** - demotion happens when a process is migrated from a higher priority to a lower priority queue. Default: A process is demoted if it cannot complete execution after it has consumed its allocation. Currently, this can only be implemented if the scheduling policy is round robin.
   c. ***Retention*** - retention happens when a process is retained in in its current queue. Default: A process is retained in a queue if no other queue with higher priority exists.
4. ***Entry policy*** - the entry policy determines which queue a new process enters. Default: Always send the process into the queue with the highest priority.
5. ***Priority policy*** - the priority policy determines how queue prioritization is implemented in the system. Possible schemes for this would be:
   a. ***Higher before lower*** - the CPU scheduler chooses a process from the queue with the highest priority. In cases where more than one process from the highest priority queue is ready, the order of execution is decided by the associated scheduling algorithm. This means that if there are 2 processes ready in the selected queue and if its associated scheduling algorithm is round robin with a time quantum of 4, the CPU will be multiplexed between these two processes.
   b. ***Fixed time slots*** - In this priority policy we allocate CPU time across the different queues proportional to its priority. E.g., suppose there are 3 queues Q1, Q2, and Q3 in the system with relative priorities 3, 2, 1, respectively. Then we can allocate CPU time across the different queues as such, 3 CPU times for Q1, followed by 2 CPU times for Q2, and 1 CPU time for Q3. The scheduler then cycles back to Q1 after it has allocated CPU time for Q3.

VII. TESTING WITH GANTT CHART

There are three samples that are shown in Table 1, 2, and 3. In each sample, there are three figures shows Gantt chart in which the first corresponds to the Non-MLFQ (using scheduling without movement between queues), second MLFQ using a priority policy of higher before lower configuration, and third MLFQ using a priority policy of fixed time slots configuration.

Process ID	Arrival Time	Burst Time	Priority
1	1	16	4
2	9	13	8
3	0	29	10

Table 1: Sample 1

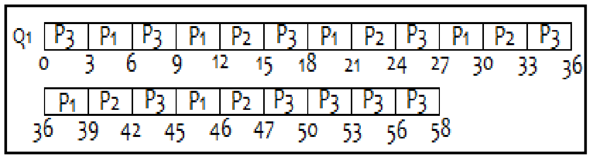

Figure 5: FCFS

Average Response Time: 1.6666
Average Waiting Time: 27.66666

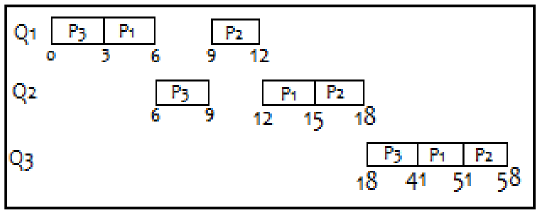

Figure 6: Higher before lower (Q1: RR time quantum: 3, Q2: RR time quantum: 3, Q3: FCFS)

Average Response Time: 0.66667
Average Waiting Time: 27.3334

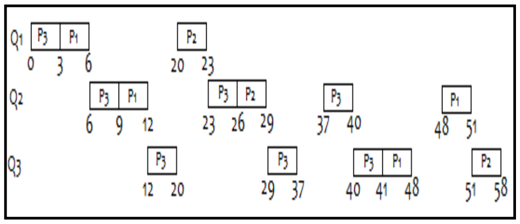

Figure 7: Fixed Time Slice (Q1: RR time quantum: 3, Q2: RR time quantum: 3, Q3: FCFS)

Average Response Time: 4.33335
Average Waiting Time: 27.3334

Process ID	Arrival Time	Burst Time	Priority
1	10	43	10
2	9	12	6
3	0	15	9

Table 2: Sample 2

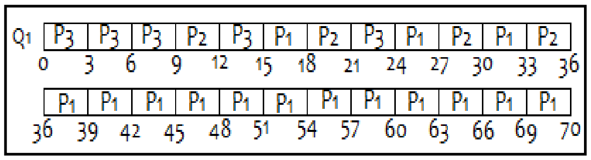

Figure 8: FCFS

Average Response Time: 1.66666
Average Waiting Time: 13.666667

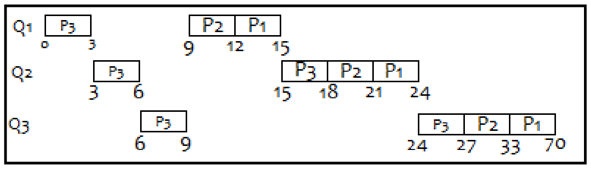

Figure 9: Higher before lower (Q1: RR time quantum: 3, Q2: RR time Quantum: 3, Q3: FCFS)

Average Response Time: 0.66667
Average Waiting Time: 13.666667

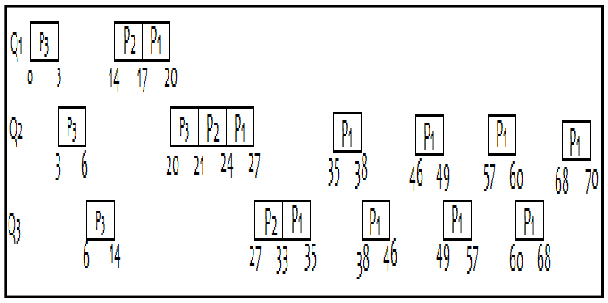

Figure 10: Fixed Time Slice (Q1: RR time quantum: 3, Q2: RR time quantum: 3, Q3 : FCFS)

Average Response Time: 4.0
Average Waiting Time: 11.666667

Process ID	Arrival Time	Burst Time	Priority
1	6	20	5
2	8	3	9
3	0	26	3

Table 3: Sample 3

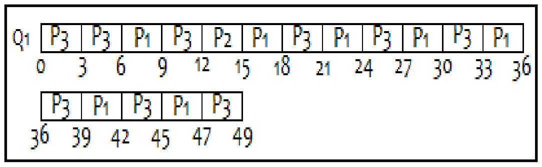

Figure 11: FCFS

Average Response Time: 1.33334
Average Waiting Time: 16.0

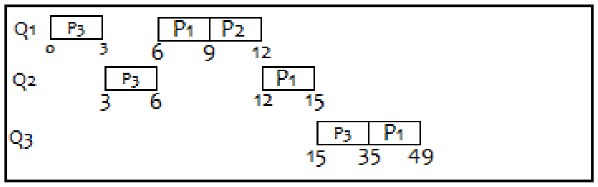

Figure 12: Higher before lower (Q1: RR time quantum: 3, Q2: RR time quantum: 3, Q3 : FCFS)

Average Response Time: 0.33334
Average Waiting Time: 11.0

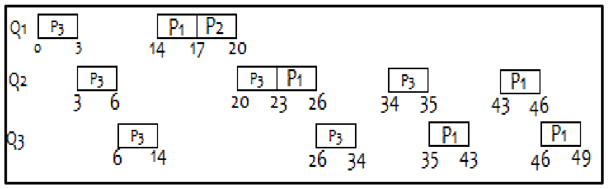

Figure 13: Fixed Time Slice(Q1: RR time quantum: 3, Q2: RR time quantum: 3, Q3 : FCFS)

Average Response Time: 5.666665
Average Waiting Time: 13.666667

VIII. COMPUTAIONAL RESULTS

In a CPU Scheduling Algorithm, the criteria for having best scheduling algorithm is to minimize the turnaround time, response time and waiting time.

Figure 14 and 15 shows the average response time and waiting time from the test samples above. The average response time is stable on the MLFQ using a priority policy of higher before lower configuration. While the average waiting time depends on the sample's time slicer quantum.

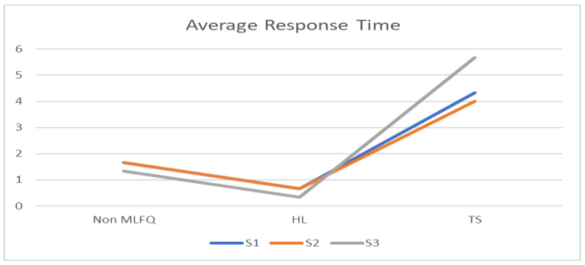

Figure 14: Average Response Time from Samples Above

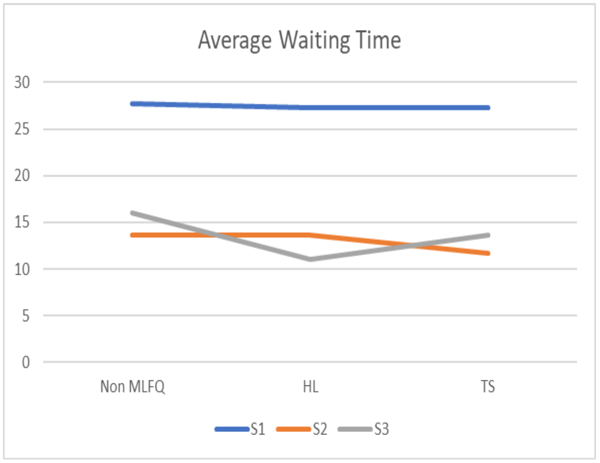

Figure 15: Average Waiting Time from Samples Above

There are eight random test samples that was conducted, and each test sample has 17 random processes. The average response time, average turnaround time and average waiting time of each test samples was solve using the processes that have lower priority among the 17 processes of each test sample.

Among the three approaches that was conducted, which are the Non-MLFQ, MLFQ using a priority policy of Higher before lower configuration, and MLFQ using a priority policy of Fixed time slots configuration, the MLFQ using a priority policy of Higher before lower configuration has the best response time among the three approaches. As Figure 16 indicated.

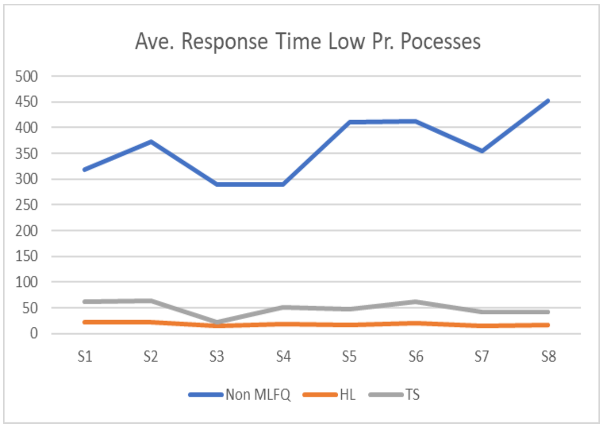

Figure 16: Average Response Time of Low Priority Processes (8 samples, 17 processes, [for HL and TS] Q1: RR 3, Q2: RR 3, Q3: PrS, [for Non-MLFQ] PrS)

In the case of the average turnaround time and the waiting, the best approach varies to the configuration of the scheduling algorithm depending on the time slicer and time quantum given.

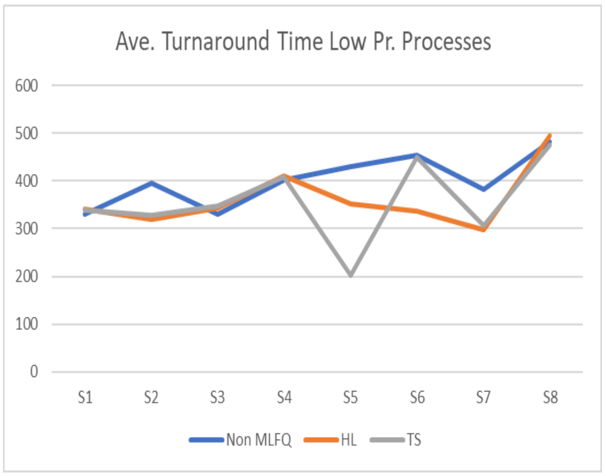

Figure 17: Average Turnaround Time of Low Priority Processes (8 samples, 17 processes, [for HL and TS] Q1: RR 3, Q2: RR 3, Q3: PrS, [for Non-MLFQ] PrS)

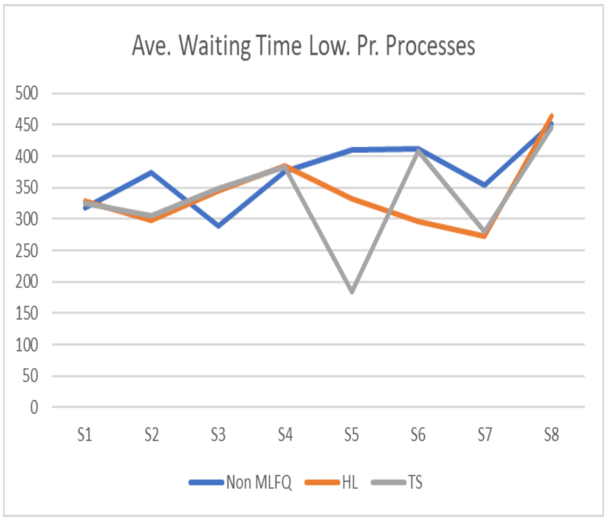

Figure 18: Average Waiting Time of Low Priority Processes (8 samples, 17 processes, [for HL and TS] Q1: RR 3, Q2: RR 3, Q3: PrS, [for Non-MLFQ] PrS)

IX. CONCLUSION

The hypothesis is true depending on the configuration that is better for that test data.

The current configuration of MLFQ with promoting and demoting processes and the option of two priority policies shows a great role in contemporary scheduling algorithm to be better in terms of scheduling processes in the CPU. As the samples shown in the computational results shows some of the good characteristics by the given priority policy where the Higher before lower configuration has the better results of average response time and the Fixed time slots configuration having the better results of average waiting time or average turnaround time.

It shows that it has decreased the problem of starvation that low priority processes can get faster response to the scheduler and smaller waiting and turnaround time.

X.REFERENCES

[1] M. V. Panduranga Rao and K. C. Shet, "Analysis of New Multi-Level Feedback Queue Scheduler for Real Time Kernel", International Journal of Computational Cognition (http://www.ijcc.us), vol.8, no. 3, September 2010.

[2] Hoganson, Kenneth (2009), "Reducing MLFQ Scheduling Starvation with Feedback and Exponential Averaging", Consortium for Computing Sciences in Colleges, Southeastern Conference, Georgia.

[3] Parvar, Mohammad R.E, Parvar, M. E. and Safari, Saeed (2008), "A Starvation Free IMLFQ Scheduling Algorithm Based on Neural Network", International Journal of Computational Intelligence Research ISSN 0973-1873 Vol.4, No.1 pp. 27–36

[4] Operating System Concepts by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne.

[5] Ayan Bhunia, "Enhancing the Performance of Feedback Scheduling" International Journal of Computer Applications (0975 – 8887), Volume 18 No.4, March 2011.

[6] Becchetti, L., Leonardi, S. and Marchetti S.A. (2006), "Average-Case and Smoothed Competitive Analysis of the Multilevel Feedback Algorithm" Mathematics of Operation Research Vol. 31, No. 1, February, pp. 85–108.

[7] Basney, Jim and Livny, Miron (2000), "Managing Network Resources in Condor", 9th IEEE Proceedings of the International Symposium on High Performance Distributed Computing, Washington, DC, USA.

[8] Scheduling: The Multilevel Feedback Queue. http://pages.cs.wisc.edu/~remzi/Classes/537/Fall2009/Notes/cpu-sched-mlfq.pdf

[9] Liu, Chang, Zhao, Zhiwen and Liu, Fang (2009), "An Insight into the Architecture of Condor –A Distributed Scheduler", IEEE, Beijing, China.

[10] Wolski, Malhar Thombare, Rajiv Sukhwani, Priyam Shah, Sheetal Chaudhari, Pooja Raundale (2016), "Efficient implementation of Multilevel Feedback Queue Scheduling" , IEEE, Sardar Patel Institute of Technology Andheri, Mumbai, India.

[11] Patel, Jyotirmay, A.K. Solanki (2012), "Performance Enhancement of CPU Scheduling by Hybrid Algorithms Using Genetic Approach", International Journal of Advanced Research in Computer Engineering & Technology ISSN: 2278 – 1323

[12] Romero, Victor, "Machine Problem 01: Multiple Feedback Queue Scheduling", 2018

[13] Aoshima, Naoto, Kimura, Haruo, Minai, Katsuhito et al (1998), "Process Scheduling System that Allocates CPU Time with Priority to Interactive Type Processes", Fujitsu Limited, Kawasaki, Japan