***Abstract -*** **Each operating system in use or being developed selects a CPU scheduling algorithm that helps to dictate the order in which processes are completed. Microsoft Windows operating system, for example, utilizes an algorithm called a multi-level feedback queue. Multi-level Feedback Queues(MLFQ) are a scheduling algorithm that uses past behavior to predict future usage and assign job priorities. When a process first begins it starts at the highest priority, then as it continues to consume CPU time without completion, its’ priority is reduced. As the process is lowered in priority, it is allowed to run for a longer time frame (quantum). MLFQ utilizes multiple priority queues in conjunction with a basic round-robin scheduling algorithm to ensure processes complete in a timely manner. In this research we examine the possibility of improving turnaround time, a key CPU scheduling algorithm metric. Turnaround time is the time it takes for a process to complete once it is ready to begin. The intention is to customize the MLFQ based on a specific individual’s computer habits by altering the time slices that each queue in the MLFQ utilizes based on the findings from the selected** **machine learning algorithm. We begin by proofing the concept with simulated datasets, then capturing live data from the researcher’s workstations, and finish by training and testing the machine learning algorithm with the captured live data.**

***Index Terms –* scheduling algorithm, multi-level feedback queue, machine learning, random forest**

1. INTRODUCTION

While varying systems do have customized scheduling algorithms to suit their purposes, generic operating systems, such as Windows, do not. The purpose of this study is to examine the feasibility of utilizing machine learning to customize a scheduling algorithm that optimizes turnaround time based on the needs of an individual user in a system such as Windows. Working under the assumption that most individuals utilize computers for a similar purpose day in and out: That is to say that the reasons end users are on the computer don’t tend to change drastically, a supervised machine learning algorithm could in theory optimize the MLFQ scheduling time quantum. As an end user spent more time utilizing the system, the closer towards being optimized it would then become.

For the purposes of this research, several controls were implemented to isolate testing solely on time quantum size in each queue. The first control is that the MLFQ algorithm will always contain four different priority queues. The second is that each subsequent queue had a time quantum that was twice the length of its’ predecessor. Finally, the largest time quantum that would be tested is equal to, or the first instance greater than, the largest burst time required by a process.

1. BACKGROUND & RELATED WORK

The previous research conducted in the area of Multi level Feedback Queue scheduling and time quantum values was mostly a comparative study of all the scheduling algorithms i.e First Come First Serve, Round Robin and Multi level Feedback Queue. It was concluded that the time quantum values of 4 and 13 were proven to be most effective in terms of response time, throughput, wait time and turn around time [1].

In yet another research paper, various machine learning algorithms were introduced to estimate the CPU burst time for different processes. Regression algorithms such as KNN, Dtree, Random Forest and XGBoost were used to build a predictive model and estimate the burst time [2]. Random Forest performed optimally among all of the algorithms considered.

With MLFQ, the factors that need to be taken into consideration as addressed in the book [3], are the number of queues, the time quantum value for each queue and if priority needs to be boosted for a particular queue. These questions thus become the crux and main idea behind the MLFQ queue optimization that this paper presents.

1. METHODOLOGY

To begin the research, three simulated datasets would be randomly generated by Python code then output to a .CSV file. The purpose of the initial datasets will be to prove that altering the time quantum for the queues can help improve turnaround time. For the initial simulated datasets, the optimization of turnaround time will be based solely on the average turnaround time of each process. This makes the results of the tests agnostic of how many processes finished in any particular queue. To isolate specifically the time quantum, there were several controls implemented.

First, for the simulated datasets we limited the number of queues that the MLFQ can utilize to four. While most real operating systems utilize more queues than that, this allows us to set a simplified baseline for testing the optimization of the time quantum. Secondly, each successive queue in initial testing will utilize a time quantum double that of its’ successor. For example, if the first time quantum slice is two milliseconds, the second queue would then have a time quantum of four milliseconds, the third would have a quantum of eight milliseconds, and so on. Meaning that each of the longest queues would then be eight times the length of the original.

Third, we began testing with the first time quantum set at one, with the algorithm then doubling the time quantum for each queue at each successive run. Finally, once the largest quantum was larger than the burst time of the longest process, the testing would cease. The corollary to this is noting that in that instance, no process added, regardless of size, would have to go through the last queue’s wait time more than once. Thus, in that instance the round robin effect on the queue would be essentially nullified and it would simply be a queue. This would also render the fifth rule of MLFQ’s irrelevant as well, as no process would then arbitrarily need to be pushed back to the first queue.

After proofing that the simulated datasets can be optimized through a different time quantum, the next step is to identify the best machine learning algorithm to be utilized for the optimization of the dataset.

Once the best ML algorithm is identified, live data from a computer will be collected. As an additional control in this step, the live data capture length will be set to a static five minutes. A solution will be coded to capture the live processes from a computer utilizing the PSUtil Python library [7]. After capturing the process data from the live computer, the output .CSV will then be cleaned for data that may skew the model.

Finally, the datasets will be sent through the machine learning model to train it based on each device’s output. After running the model with the datasets, additional python libraries, such as matplotlib and others [8][9][10][11][12][13][14], will be utilized to output the data for review. Any potential clustering, or ability to alter the time quantum on the live set will be identified as a potential point of improvement. Conclusions will be drawn from the outputted data and future work then built out.

1. RESEARCH

The research began by building out the code for the multi-level feedback queue. For this research’s purposes all code was done in Python. MLFQ algorithms begin with a base set of five rules that they follow, so the code had to make sure and accommodate that.

After implementing the controls and the MLFQ function, the program then ran the testing algorithm shown in figure 1.

time\_quantum = 1 //starting value based on control

longest\_quantum = time\_quantum \* 8

WHILE longest\_quantum < longest\_process:

RUN mlfq(time\_quantum)

CALCULATE and output TAT for each

process

CALCULATE average TAT

DOUBLE time\_quantum, longest\_quantum

COMPARE average\_turnaround\_time

Fig. 1. MLFQ Time Quantum Testing Algorithm

For initial testing and proof of concept, a simplified MLFQ code was utilized. The algorithm at that point in time did not have an ageing component or the fifth rule that arbitrarily moved every process not in the first queue back into the first queue. The first tests ran with this code were on three simulated datasets consisting of five hundred processes each.

These datasets contained a process identifier (PID), arrival time (AT), and burst time (BT). The arrival time for these simulated datasets was between zero and five milliseconds. The initial burst time was a randomly generated number between one and ten milliseconds. The waiting time and turnaround time were then calculated through the program run.

During this initial testing, it quickly became apparent that further tests would need to be conducted with expanded data. Due to the high number of processes, namely five hundred, hitting the queue all within a five-millisecond burst time, the turnaround time on many of the processes exceeded 1000 milliseconds. There also needed to be an expansion on the size of the burst times for the processes, as the upper bound of ten milliseconds only allowed for two tests to be conducted. The first test with one, two, four, and eight millisecond queues; and the second with two, four, eight, and sixteen millisecond queues. The results were then inconclusive as the testing dataset was not optimal.

After expanding the datasets with a wider range of burst times and a larger range of arrival times, better data emerged. The results successfully indicated that for the three artificial datasets tested, a different time quantum was optimal for one of them, the other two had the same quantum for the optimal average turnaround time. To this point, the work completed was similar to that of [1]. That research was able to conclude a very similar point, thus this was just a proof of their work to build upon.

Next, the machine learning algorithm best suited to the needs of the experiment was chosen. It has been shown that the random forest algorithm is best suited to the optimization of this kind of dataset. The random forests are flexible ensemble classifiers based on decision trees. They construct multiple trees based on random bootstrapped samples of the training dataset. This would be utilized to sift the training data from the live dataset to find groupings.

To capture the live dataset from each machine. The original intention was to utilize the PSUtil Python library. However, after exploring the library and building a program to output the list of running processes, it was unable to capture the burst time of each process in execution. After further research, it is found that data is not accessible to end users on the Windows operating system.

While searching for a workaround to this problem, another software already in production was identified to capture the live processes. Windows Process Monitor provided an easy-to-use interface that captured all the same data as the PSUtil library, but simplified the ability to capture data for a set amount of time. However, once again that program still did not gather burst time, although it did gather start, end, user, and kernel time spent by a process.

To supplement the lack of a provided burst time to research with, a randomized field was added to the data. This randomized field was a number between zero and the length of total time taken by the process. After cleaning the data, the input files retained the following fields:

1. Time of day – when process was captured.
2. Process Name
3. Process ID (PID)
4. Operation – what the process was doing.
5. User or Kernel time
6. Kernel time
7. Total time (in milliseconds) – calculated from the combination of column 5 and 6.
8. Burst time (randomly generated) – random number between zero and the total in column

When cleaning the data, some processes did not have user time. In this instance, the time in column five was solely kernel time. If there was user time by the process, it was then in column five and kernel time was pushed to column six. Because the differentiator between user and kernel time was not the purpose of this research, the columns were simply totaled.

After capturing the live data from our three workstations, during the cleaning process, the decision was made to discard one of the computer’s data. Despite the time frame being the same, one of the computer’s dataset was approximately three times larger than the other two. That computer captured over one hundred thousand processes running in those five minutes, the other two captured just over thirty thousand. The decision was made to treat that computer’s data as an outlier, as it could have been damaging to the training set of the random forest algorithm. To supplement the loss of the third computer’s data, another artificial dataset was created, this time mirroring the data captured in a live environment.

Once the datasets were prepped, they were then fed through the random forest algorithm to find potential patterns.

The random forest algorithm did not implement the same controls as the simulated data. The queues themselves weren’t limited to four and the doubling component was removed. The intention for the algorithm was to simply work on predicting size of processes incoming to see if patterns emerged that could potentially be utilized.

Looking over the output from the supplemented, artificial dataset, interestingly no patterns emerged. As figure two demonstrates, the processes had no grouping.

A picture containing text, screenshot, line, receipt

Description automatically generated

Fig. 2. Time Taken by Process from Simulated Dataset

This can be further demonstrated by looking over the frequency of predicted quantum within the dataset. Figure three shows the top five frequencies identified from the simulated dataset. The highest frequency of predicted time quantum is two and the prediction equates to approximately four seconds.

A picture containing text, receipt, screenshot, font

Description automatically generated

Fig. 3. Frequency of Predicted Time Quantum

However, as the focus moves from the artificial dataset to the real datasets from the two workstations, patterns begin to emerge. Looking over the Gantt chart from figure four and five, groupings emerge in the time taken. There’s also a major gap between processes that can finish quickly, as both datasets have a far shorter turnaround time.

A picture containing text, screenshot, display, number

Description automatically generated

Fig. 4. Time Taken by Process from 1st Live Dataset

A picture containing text, screenshot, diagram, line

Description automatically generated

Fig. 5. Time Taken by Process from 2nd Live Dataset

In addition to the shorter turnaround time, they also have a much more prominent grouping in frequency of predicted times, as shown in figure six and seven.

Both datasets have sizable groupings that emerged with very similar frequencies and predicted time quantum. For the first queue, a predicted time of 15.625 and 31.25 milliseconds appeared frequently. Interestingly, despite not forcing the random forest algorithm to double the time quantum in successive trials, it found that the doubled quantum in the first queue was the most effective prediction. Furthermore, the top five frequencies across the datasets were the same quantum, although with different predicted frequencies.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Queue Index | Predicted Time Quantum | | Frequency |  |
| 0 | 15.625 | 81 | |  |
| 0 | 31.25 | 49 | |  |
| 1 | 46.875 | 25 | |  |
| 1 | 62.5 | 17 | |  |
| 1 | 78.125 | 15 | |  |

Fig. 6. Predicted Time Quantum for 1st Live Dataset

|  |  |  |
| --- | --- | --- |
| Queue Index | Predicted Time Quantum | Frequency |
| 0 | 15.625 | 110 |
| 0 | 31.25 | 33 |
| 1 | 46.875 | 11 |
| 1 | 78.125 | 11 |
| 1 | 62.5 | 9 |

Fig. 7. Predicted Time Quantum for 2nd Live Dataset

Because of these groupings, the turnaround time on the live datasets was far improved over the simulated artificial dataset as shown in figure eight, nine, and ten.

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

Fig. 8. Turnaround Time by Queue for Artificial Dataset

A picture containing text, screenshot, diagram, rectangle

Description automatically generated Fig. 9. Turnaround Time by Queue for 1st Live Dataset

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

Fig. 10. Turnaround Time by Queue for 2nd Live Dataset

1. CONCLUSION

In this study, we aimed to predict time quantum based on personalized computer use by employing machine learning techniques in the Multi-Level Feedback Queue (MLFQ) scheduling algorithm. Our research utilized three distinct datasets, which captured diverse user profiles, hardware configurations, and software environments, to train and evaluate the performance of our prediction model.

The results of our analyses demonstrated that the machine learning-based model significantly improved the accuracy of time quantum prediction, compared to the traditional MLFQ approach. This optimization allowed for better resource allocation, reduced waiting times, and increased overall system performance. Furthermore, the model was adaptable to various user behaviours and computing environments, emphasizing its practical applicability and potential for widespread adoption.

In conclusion, the integration of machine learning techniques within the MLFQ scheduling algorithm has proven to be a promising approach for predicting time quantum based on personalized computer use. Our findings suggest that the proposed model can lead to substantial improvements in system performance and user satisfaction by optimizing resource management and adapting to diverse computing scenarios. This study opens the door for further exploration and refinement of machine learning-based models in the domain of computer system scheduling and resource allocation.

1. LIMITATIONS & FUTURE WORK

Limitations

Despite the promising results obtained in this study, there are several limitations that should be acknowledged:

1. Limited Datasets: Although we used three diverse datasets, they may not fully capture the wide range of user behaviors and computing environments that exist in the real world. As a result, the model's performance may not generalize well to other scenarios not represented in the datasets.
2. Model Complexity: The machine learning techniques employed in our study may introduce additional computational overhead, potentially offsetting some of the performance gains achieved through improved time quantum prediction.
3. Static Features: The current model assumes that the features used for prediction remain relatively static over time. However, this may not be the case, as users can change their behavior, and hardware and software components can be updated or replaced.

Future Scope

Considering the limitations mentioned above, there are several directions for future research:

1. Expanding Datasets: To improve the model's generalizability, it would be beneficial to collect and utilize more diverse datasets, covering a broader range of user behaviors, hardware configurations, and software environments.
2. Model Refinement: Investigating alternative machine learning techniques, such as deep learning or ensemble methods, could potentially lead to better performance and more accurate time quantum predictions.
3. Dynamic Adaptation: Developing models that can dynamically adapt to changing user behaviors and computing environments would be valuable. This may involve incorporating online learning techniques, which update the model as new data becomes available.
4. Real-World Deployment: Implementing the proposed model in real-world computer systems would help to evaluate its practical applicability and effectiveness in a variety of contexts. This could also provide insights into any unforeseen challenges or performance issues that may arise during real-world usage.
5. Comparative Analysis: Performing a comprehensive comparison of the proposed model with other scheduling algorithms and resource allocation techniques would provide a better understanding of its relative advantages and limitations.
6. BIBLIOGRAPHY & REFERENCES
7. D. Mulder et al., "Impact of Quantum Values on Multilevel Feedback Queue for CPU Scheduling," 2020 Intermountain Engineering, Technology and Computing (IETC), Orem, UT, USA, 2020, pp. 1-4, doi: 10.1109/IETC47856.2020.9249188.
8. P. Samal, S. Jha and R. K. Goyal, "CPU Burst-Time Estimation using Machine Learning," *2022 IEEE Delhi Section Conference (DELCON)*, New Delhi, India, 2022, pp. 1-6, doi: 10.1109/DELCON54057.2022.9753639.
9. R Arbaci-Dusseau and A. Arpaci-Dusseau, “Scheduling: The Multi-Level Feedback Queue,” in *Operating Systems: Three Easy Concepts, 1st ed. 2013,* ch. *8, pp. 1-10*.
10. L. Breiman, “Random Forests,” *Machine Learning,* vol. 45, pp. 5-32, 2001, doi:10.1023/A:1010933404324
11. S. A. Naghibi, K. Ahmadi, and A. Daneshi, “Application of Support Vector Machine, Random Forest, and Genetic Algorithm Optimized Random Forest Models in Groundwater Potential Mapping,” *Water Resources Management,* vol. *31,* no. *9*, pp. 2761-2775. July 2017, doi: 10.1007/s11269-017-1660-3
12. M. Alsanea, “Machine learning methods for CPU scheduling,” *Trans. On ML and Data Mining,* vol. *14,* no. *2,* 2021, P-ISSN: 1865-6781, E-ISSlN 2509-9337, ISBN 978-3-942952-89-7

Python Libraries

[7] <https://pypi.org/project/psutil/>

[8] <https://pandas.pydata.org/>

[9] <https://numpy.org/>

[10] <https://seaborn.pydata.org/>

[11] <https://matplotlib.org/>

[12] <https://pypi.org/project/tabulate/>

[13] <https://docs.python.org/3/library/random.html>

[14]  [https://docs.python.org/3/library/datetime.html](%20https://docs.python.org/3/library/datetime.html)