Implementing Machine Learning to Optimize CPU Task Scheduling

Danh Pham

Georgia State University

Department of Computer Sciecne

Atlanta, GA 30302

dpham16@student.gsu.edu

***Abstract*— Since the creation of the Operating System (OS), computer engineers have been designing a variety of CPU scheduling algorithms to optimize the execution of processes for the I/O and CPU. The existing research background on CPU task scheduling right now is based on the long-term scheduler, mid-term scheduler, and short-term scheduler. The overall reaching goal is to create or select an efficient algorithm that minimizes cost of operations and performance time while maximizing hardware usage.**

Keywords—Machine Learning; CPU Task Scheduling; Operating System; Scheduling Algorithms; Neural Networks; Artificial Intelligence

INTRODUCTION

In the contemporary computer science field, scheduling is the main foundation of how operating systems allocate resources for computer usage. Operating systems currently follow some certain guidelines on how the process scheduling task is conducted for the maximum use of the CPU for a system. The existing research background on CPU task scheduling right now is based on the long-term scheduler, mid-term scheduler, and short-term scheduler.

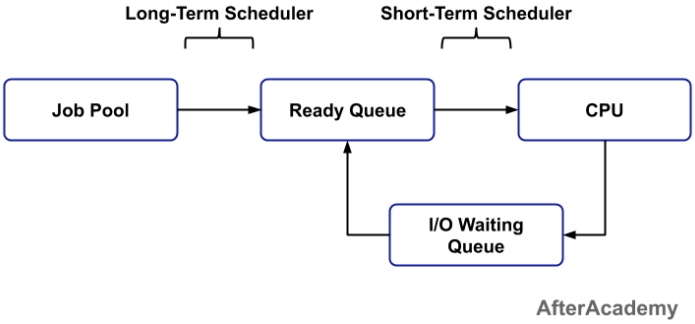

Figure 1: *Here we have a figure overview of the different types of schedulers and the flow of their operations.*

A. Long Term Scheduling

To start off, the long-term scheduler is in charge of setting processes in the ready queue for CPU allocation. Whenever there is a program to be executed, the long-term scheduler would be the main decider of allocating whichever task goes into the ready queue. Currently, long-term scheduling is fundamental to many industry-scale technology centers that require extensive computations.

B. Medium Term Scheduling

Furthermore, the medium-term scheduler is in charge of clearing away processes that are in the computer's main memory, which is the RAM of the computer, and switches those processes into the computer's secondary memory. Currently operating systems have determined that medium term scheduling is primarily used for computers to alleviate the burden of multi-programming.

C. Short Term Scheduling

Lastly, the short-term scheduler is in charge of handling how the CPU behaves after certain system interrupts, such as an I/O interrupt, calls from the operating systems signaling, or other types of calls. When these types of calls happen, the short-term scheduler has to decide which process inside the ready queue will be executed for the computer's scheduling to commence

II. RELATED WORK

Currently engineers have based these three types of foundational scheduling concepts of long-term scheduling, medium term-scheduling, and short-term scheduling into developing a guideline of scheduling algorithms that have been useful since their conception. By applying these concepts, researchers designed scheduling algorithms such as First Come First Serve (FCFS), Shortest Job First (SFJ), Longest Job First (LJF), Priority Scheduling (PS), Shortest Remaining Time First (SRTF), Round Robin (RR), and more [1]. Each scheduling algorithm has their strengths and weaknesses. For example, Shortest Job First (SJF) and Shortest Remaining Time are favorable for processes that would only require a short burst length; However, these algorithms are not optimal for longer processes because it would starve longer processes in cases where there are a high number of short processes that need to be executed. It is very beneficial to obtain a further understanding of the scheduling algorithms researchers have proposed over the years.

A. First Come First Serve

To start off, First Come First Serve, which is commonly abbreviated as (FCFS), is considered to be one of the most straightforward scheduling concepts designed today. This type of algorithm determines the execution of a task, on the order which a process arrives. First Come First Serve suffers from not being able to prioritize a task based on their weight of execution because it relies purely on which task the comes first for the operating system.

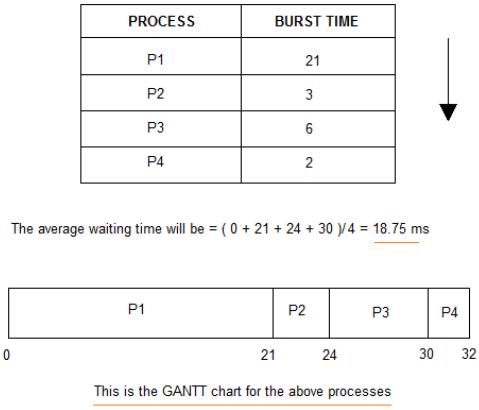

Figure 2: *This figure shows a typical First Come First Serve process, burst time, and Gantt chart that is typically performed as the computation inside the CPU.*

B. *Shortest Job First*

Furthermore, Shortest Job First, which is commonly abbreviated as (SJF), is another straightforward scheduling algorithm that is commonly used. The Shortest Job First algorithm can come into two different types of forms: non-preemptive shortest job first or preemptive shortest job first; However, both of these variants follow the same principal of choosing the process that comes into the operating system which has the minimal amount of execution time amongst all the other processes that arrived at the CPU.

C. *Longest Job First*

To add on, Longest Job First, which is commonly abbreviated as (LJF), is another simple algorithm that researchers have developed for OS CPU task scheduling. This concept of scheduling also comes into two variants: non-preemptive longest job first or preemptive longest job first. Basically, the longest job first algorithm is the opposite of shortest job first. This type of algorithm typically chooses the process that arrives in the queue which the CPU determines to have the longest execution time for the CPU to work on.

D. *Shortest Remaining Time First*

To move onto more complex algorithms, we currently have the Shortest Remaining Time First (SRTF) algorithm that is currently used today. Basically, this algorithm is a preemptive way which the operating system would determine the execution of a task primarily based on if the job is near completion. This type of algorithm is more complex because the computer has to make predictions on a task's completion time and use that prediction as a basis for how the operating system prioritizes that certain task for execution.

E. *Round Robin*

In contrast, a more straightforward scheduling algorithm that has been thoroughly developed, is the Round Robin (RR) concept of CPU task scheduling. The round robin primarily focuses on the concept of passing around the tasks in the concept of a cycle for the CPU to complete that task in a pre-determined established amount of time, where if the task is not finished in that time, then it wouldn't be executed; However, it would re-enter that cycle, which would postpone that task, but it would also solves the problem of starvation for the processes in the operating system.

To further simplify, long-term scheduling is in charge of setting processes in the ready queue for CPU allocation, medium-term scheduling is responsible for swapping processes of the main memory and second memory, and short-term scheduling is responsible for executing processes in the ready queue while utilizing the clock interrupt, I/O interrupt, and OS call..

III. HYPOTHESIS

A feasible and initial new method to create an optimal CPU task scheduling algorithm is to incorporate machine learning for task scheduling. Machine Learning has already been used to train computers to optimize efficiency in other engineering fields, and these results could be applied for CPU scheduling algorithms which would optimize the CPU's utilization, turnaround time, response time, waiting time, and throughput. [2] Basically, it would be possible to implement a neural net algorithm that would determine the best scheduling algorithm for optimal process burst time, and this would create an efficient command point for the operating system.

The current research plan is to apply a sample of scheduling algorithms, such as shortest job first and first come first serve as competitors. They are assigned process ID's and randomized burst times. When fed the randomized burst time as input layers, the neural net process this information and predict which scheduling algorithm would win the next assignment of scheduling. In the future, we would work on real operating system burst time data when performing task and use that information as the predictor of which scheduling algorithm would win.

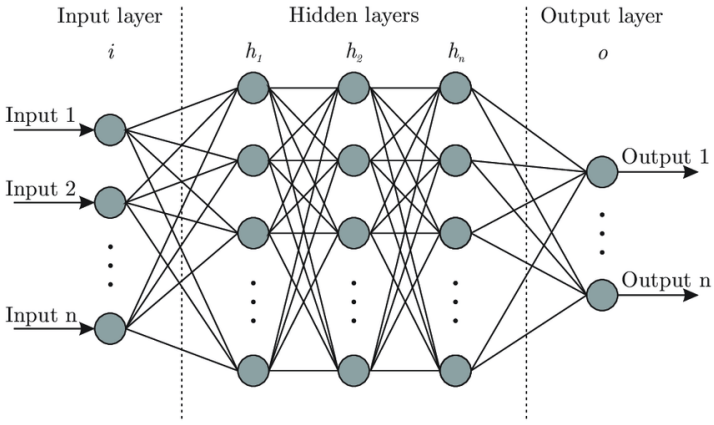

Figure 3: *Our input layer would be our initial dataset. The hidden layers would be the training calculation which refines the weight and cost. Our output layer would be the results which we propagate backwards from.*

IV. EXPERIMENT

In this experiment, I decided to use non-preemptive First Come First Serve and preemptive Shortest Job First as the competing algorithms. The algorithms were fed randomized process burst time ranging from 1ms to 100ms for calculations.

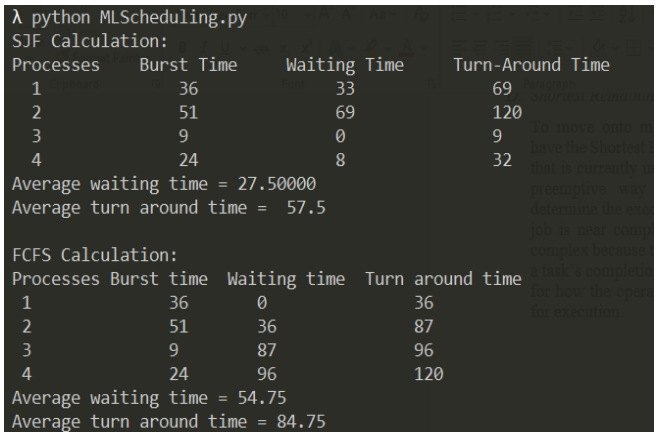

Figure 4: *Gather algorithm results*

Subsequently, the average waiting time and turnaround time would be used as data for the neural network to predict the next winner of scheduling. The dataset was normalized by averaging and scaling the inputs to 0 or 1. The data is then trained through a neural net of sigmoid and synaptic weight calculations, which is repeatedly iterated through 1000 times to obtain the output layer and prediction value. Finally, the predicted winner would be the one that is closest to the prediction value score after the process finishes.

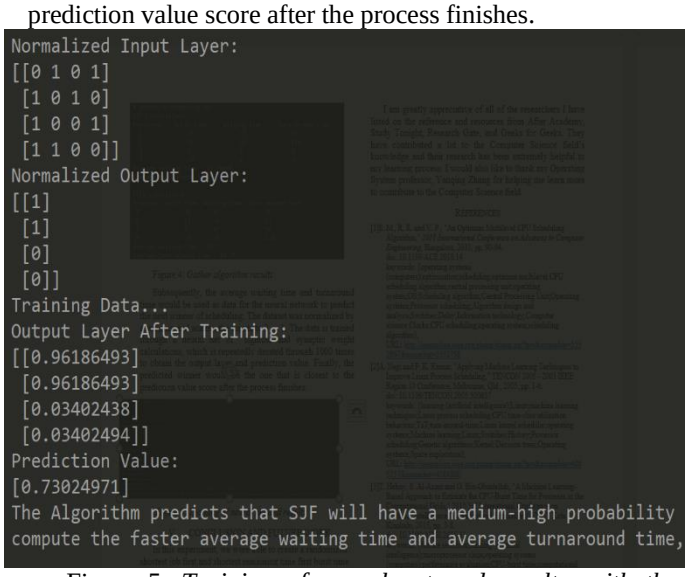

Figure 5: *Training of neural net and results, with the algorithm predicting that SJF will have a medium-high probability of outperforming FCFS in the next scheduling process.*

V. CONCLUSION AND FUTURE WORK

In this experiment, we were able to create a randomized shortest job first and shortest remaining time first burst time pseudo-algorithm, which would feed into our neural network as data. By using this data, the neural network would predict the chances of which scheduling algorithm would win over the other one when performing process scheduling. We found out that the SJF algorithm will most likely perform better than FCFS when the burst times are reasonably low and when there is not a lot of processes.

For the future plan, we would implement our previous algorithm into real operating systems that would gather data when the system is performing tasks. The training data would incorporate performance testing which would factor in the successes and failures in the data subset for the machine to find optimization. We would utilize the training dataset for further validation testing while filtering out noise from the previous dataset to refine our data. The datasets would record the length of CPU bursts, throughput, turnaround time, response time, waiting time, and utilization, etc. From the provided supervised learning training set and further validation testing, it would be possible to create a machine learning decision tree which would design a more optimal CPU task scheduling algorithm for higher levels of efficiency[3].

ACKNOWLEDGMENTS

I am greatly appreciative of all of the researchers I have listed on the reference and resources from After Academy, Study Tonight, Research Gate, and Geeks for Geeks. They have contributed a lot to the Computer Science field's knowledge and their research has been extremely helpful to my learning process. I would also like to thank my Operating System professor, Yanqing Zhang for helping me learn more to contribute to the Computer Science field.

REFERENCES

[1]S. M., R. R. and V. P., "An Optimum Multilevel CPU Scheduling Algorithm," *2010 International Conference on Advances in Computer Engineering*, Bangalore, 2010, pp. 90-94.
doi: 10.1109/ACE.2010.14
keywords: {operating systems (computers);optimisation;scheduling;optimum multilevel CPU scheduling algorithm;central processing unit;operating system;OS;Scheduling algorithm;Central Processing Unit;Operating systems;Processor scheduling;Algorithm design and analysis;Switches;Delay;Information technology;Computer science;Clocks;CPU scheduling;operating system;scheduling algorithm},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5532867&isnumber=5532798

[2]A. Negi and P. K. Kumar, "Applying Machine Learning Techniques to Improve Linux Process Scheduling," TENCON 2005 - 2005 IEEE Region 10 Conference, Melbourne, Qld., 2005, pp. 1-6.
doi: 10.1109/TENCON.2005.300837
keywords: {learning (artificial intelligence);Linux;machine learning techniques;Linux process scheduling;CPU time-slice utilization behaviour;TaT;turn-around-time;Linux kernel scheduler;operating systems;Machine learning;Linux;Switches;History;Processor scheduling;Genetic algorithms;Kernel;Decision trees;Operating systems;Space exploration},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4085157&isnumber=4084860

[3]T. Helmy, S. Al-Azani and O. Bin-Obaidellah, "A Machine Learning-Based Approach to Estimate the CPU-Burst Time for Processes in the Computational Grids," 2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS), Kota Kinabalu, 2015, pp. 3-8.
doi: 10.1109/AIMS.2015.11
keywords: {feature selection;learning (artificial intelligence);microprocessor chips;operating systems (computers);performance evaluation;CPU-burst time;computational grids;CPU-scheduling algorithms;ML-based approach;machine learning-based approach;feature selection;grid workload dataset;GWA-T-4 AuverGrid;strength linear relationship;burst CPU time;K-NN;K-nearest neighbors;Scheduling algorithms;Computational modeling;Single machine scheduling;Testing;Artificial intelligence;CPU-Burst;CPU Scheduling Algorithm;Machine Learning;Feature Selection},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7604542&isnumber=7604529

[4]A. Kaur and B. S. Khehra, "CPU task scheduling using genetic algorithm," 2015 IEEE 3rd International Conference on MOOCs, Innovation and Technology in Education (MITE), Amritsar, 2015, pp. 66-71.
doi: 10.1109/MITE.2015.7375290
keywords: {computational complexity;genetic algorithms;microprocessor chips;optimisation;processor scheduling;CPU task scheduling;genetic algorithm;p-processes single

processor scheduling problem;total execution time;combinatorial optimization problem;NP-hard problem;meta-heuristic approach;Genetic algorithms;Processor scheduling;Job shop scheduling;Biological cells;Sociology;Statistics;CPU;scheduling;single processor;combinatorial optimization;NP-Hard;earliness;tardiness;deadline;genetic algorithm}, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7375290&isnumber=7375274

[5]Bex, P.J.G.I. "Implementing A Process Scheduler Using Neural Network Technology". Undefined, 2008, p. . https://www.semanticscholar.org/paper/Implementing-a-Process-Scheduler-Using-Neural-Bex/8d7352c3063fa079eaf0b3b5ae4e990ae4eddaa1. Accessed 16 Apr 2020