Summary of the research paper

- Titled: "**A Machine Learning Approach for Predicting Efficient CPU Scheduling Algorithm"**
- Published in 2023, 5th International Conference on Sustainable Technologies for Industry 5.0 (STI).

- Authors: Swapnil Biswas, Md. Shakil Ahmed, Md. Jobayer Rahman, Anika Khaer, Md. Motaharul Islam.
- Source: IEEE Xplore.

Summary by : Wanniarachchi C P – IT23184176
Somachandra K.P.D.M – IT23199194
Wimalasena D.C.B.V.D.S – IT23150652
Sandesh O.M.S – IT23314924

Introduction and Significance in the field of operating systems.

CPU scheduling is an important mechanism that ensures the management of system resources like CPU time, memory, and bandwidth efficiently. Multiple processes in the ready queue compete to enter the running state because a single CPU core can handle only one process at a time. So by properly scheduling these processes minimize waiting time, improve turnaround time, and maximize CPU utilization.

In traditional static scheduling, once a scheduling algorithm is selected, it remains fixed with no adaptations to process behavioural changes and increasingly complex real-time systems. Also, each algorithm works best in different situations. This results in inefficient resource allocation and high response times since they struggle with dynamic process loads. Also, some of those traditional scheduling methods can result in starvation, the convoy effect.

So the researchers have explored machine learning to create an adaptive, efficient CPU scheduling mechanism to overcome these limitations of static scheduling. By analyzing the behavior of process attributes like previous arrival time, burst time, priority, and time quantum, and machine learning models like Stochastic Gradient Descent (SGD), Logistic Regression (LR), and Support Vector Machines (SVM), their goal is to dynamically select the optimal scheduling algorithm for each process.

So, from this technique, they are trying to enhance the CPU efficiency, reduce idle time, and make many other major developments to a smart, efficient, responsive operating system.

The addressed problem and its importance

Scheduling is very important to operating systems to determine which process to run when there are multiple processes in the ready queue. It also involves allocating processor time to various processes running concurrently on the system. Also, it affects how well resources are used and how well other performance parameters like turnaround time, response time, throughput, and CPU utilization, etc, are met. Choosing the best CPU scheduling algorithm is a tough job, especially in environments with dynamic and unpredictable process behaviors. Traditional scheduling algorithms—such as First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling—operate based on fixed rules and assumptions about workload characteristics. These static algorithms are not adaptive to complex, multitasking environments where burst time, priority, and arrival time, and especially the arrival rates, frequently change.

So those traditional techniques can cause high average waiting time, more context switchings, low CPU throughput, and finally a bad user experience. This problem is more highlighted in modern systems where concurrent and diverse workloads are present in distributed and virtual environments. Time guarantee is a must in real-time systems. But the traditional algorithms struggle to meet that requirement when they start to receive fluctuating workloads. Also, maintaining energy efficiency and optimal resource utilization is critical for edge computing and cloud-based infrastructures. So they demand adaptive scheduling techniques over the traditional static methods.

Even though they have not mentioned it, we found that modern systems employ Multilevel Queue Systems and Multilevel Feedback Queues to address these static problems. MLFQ moves processes between queues based on their CPU usage. But inside each queue scheduling algorithm is static and uniform. So, processes with different needs in a queue get executed in the same algorithm. Even though there is a queue-level adaptability in MLFQ, there is still an issue with flexibility inside queues. No deep consideration of process features. It degrades performance for diverse workloads. Also, moving between queues results in overhead and delay. So these are the problems they addressed in their research.

Overview of the solution proposed by the authors.

They have tested three machine learning models for the prediction. For a fair model comparison, they have used the same features, evaluation metrics, and problem formulation. All models are linear models with low-latency prediction, making it easy to deploy in the kernel. Even though the models share a unified workflow, such as supervised learning, linear decision-making, and real-time focus, they differ in optimization goals & performance. They have implemented their proposed model in the ready queue, where the process remains waiting for execution, to decide which algorithm is best for that process. If the queue is full, the newer process will make a request, and from that moment ML model will analyze that request based on the process features.

Stochastic Gradient Descent(SGD) is a machine learning method that improves accuracy by updating model parameters with each small batch of data. In here, SGD does the predictions based on process-specific features such as burst time, previous arrival time, priority, and time quantum. During the model training phase, each sample represents a process with those features as inputs. The appropriate CPU scheduling algorithm for that process is the targeted output. Initially, the model makes predictions & calculates the error using the Mean Squared Error. Then, it adjusts the model's parameters using gradient descent to minimize the error. That will be done using the formula [$\theta = \theta - \eta . \nabla_\theta J\left(\theta; x^{(i)} : y^{(i)}\right)$]. That process is iterated to improve the model's accuracy. It will look at a different dataset with each iteration.

Then they evaluated Logistic Regression (LR), which is used to find relationships between one or more input features (independent variables) and a predicted outcome (dependent variable). The inputs and outputs are the same as the previous model. It uses the logistic regression formula,$\ln\left[\frac{\hat{p}}{1-\hat{p}}\right] = b_0 + b_1 X_1 + b_2 X_2 + \cdots + b_p X_p$ to calculate the probability that a given algorithm is the best fit for a particular process. Those probabilities were learned during the training phase from training data, and the model adjusted its coefficients to improve its ability to correctly predict the best algorithm based on patterns in the data. After training the model, it outputs a probability between 0 & 1. If the outputted probability value is greater than 0.5(selected probability threshold value), then that CPU scheduling algorithm becomes the best for that input process's features. To ensure fair learning, they normalized the input values, preventing features

with larger numerical ranges from skewing the results. This helped the model make balanced, data-driven predictions for CPU scheduling in different scenarios.

The third model they evaluated was Support Vector Machine (SVM).In here, they classify the categories (CPU scheduling algorithms) based on the process features. The SVM goal is to find the best boundary (aka hyperplane) that separates the scheduling algorithms as clearly as possible. The higher the margin between the classes and the closer the data points(aka support vectors) would be great. In their scenario, the SVM model was trained to compare two scheduling algorithms at a time. It assigns those two with labels of 1 and -1. Then it finds the optimal boundary that can classify which algorithm is best for the given process. Then the model adjusts its internal weights (weightage given to each process feature) using a formula that works with two goals. One is maximizing the margin to improve generalization and performance on new data, and the other to minimizing classification errors by controlling how far misclassified points are deviated from where they should be. For that, they use a penalty term ζ_i and a regularization parameter C in this formula, $[min_{\omega,b,\varsigma} \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\varsigma_i]$. So, each data point (a unique process with its associated feature) should be correctly classified and at least 1 unit away from the boundary. If needed to separate complex data, the model can also transform process features into a higher-dimensional space using a function $\varphi(x_i)$ in the formula, $[y_i(\omega^t\phi(x_i) + b) \geq 1 - \zeta_i]$. It helps draw more accurate boundaries when simple linear separation isn't enough. During training, the model learned from the data by updating weights, making small adjustments for correct predictions and large ones for mistakes. After training, this model can effectively predict the best scheduling algorithm for new incoming processes by accurately classifying them based on their features.

The results and their implications for the field

Each of the three models has been tested using standard performance metrics such as accuracy, mean absolute error, and root mean squared error. So among the models, SVM has shown the highest accuracy, 94.56%, where the LR shows 90.19% and SGD shows 87.26%. So, from these test results, they suggested SVMs because of its ability to classify using optimal decision boundaries for better prediction of the most suitable scheduling algorithm for a process, even though its speed and interpretability are comparatively lower than the others. Also, SVM has demonstrated lower error rates, ensuring its reliability for real-time OS environments. Even though the SGD model is faster and computationally efficient due to its incremental updates, it wastes some accuracy. It requires more iterations to converge. But it showed its potential to be efficiently deployed in lightweight systems where processing power is limited. LR model provides a good balance between interpretability and performance, making it suitable for environments where a clear explanation is needed.

So these results replace the use of traditional fixed rule scheduling algorithms and equip them to handle the dynamic environment. These machine learning based methods are integrated into the kernel layer by allowing adaptive scheduling that responds in real time to varying process loads. So the waiting time, turnaround time can be reduced, and better CPU utilization, a good user experience can be achieved. It minimizes unnecessary context switches and increases utilization by adapting to real-time workloads. Since these models are lightweight and efficient, they can be deployed in live systems without significant overhead.

As we mentioned in the “The addressed problem and its importance” section, these ML models can switch scheduling algorithms in mid-execution based on real-time process attributes. Because of that, these ML models are more accurate than modern systems that use Multilevel Queue and Multilevel Feedback Queue.

Critical evaluation of the paper

This paper evaluated the application of machine learning to optimize the CPU scheduling process with a 94.56% accuracy using Support Vector Machines. Through that, they tried to enhance scheduling efficiency while aligning with the Industry 5.0 framework. They have done a comparative evaluation and use of performance metrics like Mean Absolute Error and Root Mean Squared Error, and visualizations such as Receiver Operating Characteristic Curve and confusion matrices.

Traditional priority-based or SJF scheduling can cause lower-priority processes to starve. So, through an ML-driven scheduler, algorithms can be chosen to prevent starvation dynamically. Also, ML predictions can more intelligently decide when to preempt a process by also improving responsiveness. Also, it can minimize unnecessary context switches, so the idle time will be low. These features of the ML-driven scheduling can improve its ability to support real-time needs, where meeting deadlines and minimizing latency are critical. Because of that, ML-driven approach enhances scheduling efficiency while reinforcing key operating system principles like fairness to every process, real-time responsiveness, and balancing system resources.

A primary weakness we noticed is the isolated use of ML models, which they tested independently. They could go for a hybrid approach where stacking SGD for feature selection, LR for probability calibration, and SVM for final classification. From that, they can enhance the performance of the scheduling process.

Another thing we noticed is the smaller number of datasets used for training the models. So they should expand and diversify the dataset used during training. Also, the real-world deployment considerations, such as computational overhead, latency, and integration to the OS kernel, have not been discussed. E.g., ML models are typically written in Python and need extra libraries to function. But Python can't run inside the kernel. So they may need to convert the ML model into

C and optimize it for better integration. So they can prove the scalability and practicality of their solution in live systems.

Current models are trained at once. But in real systems, constant changes are happening. So, as an improvement, online learning can make the model keep learning from new data in real-time. So it can adapt to changes. Also, the selected model can be improved with Dynamic Voltage and Frequency Scaling to adjust the CPU power use based on demand. So the energy can be saved while still maintaining the performance well.

Since nowadays systems run across many types of devices such as mobile phones & cloud servers, this model should be tested on those platforms too. It makes sure this model works well in more complex, real-world environments. In the future, it could be integrated with advanced AI systems, such as implementing an AI agent that monitors and controls overall system performance, energy consumption, and thereby schedules tasks automatically.