Quest Journals
Journal of Software Engineering and Simulation
Volume 9 ~ Issue 7 (2023) pp: 20-29
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

ML Engine for Linux Scheduler

Anusha G H
Dept. of Electronics and Communication,
BMS College of Engineering, Bengaluru, India

Minal K
Dept. of Electronics and Communication,
BMS College of Engineering, Bengaluru, India

B P Varsha
Dept. of Electronics and Communication,
BMS College of Engineering, Bengaluru, India

Dr. Geethishree Mishra
Assistant Professor, Dept. of Electronics and Communication,
BMS College of Engineering, Bengaluru, India

Meghana G K
Dept. of Electronics and Communication,
BMS College of Engineering, Bangalore, India

Dr. Madhusudhan K. N
Assistant Professor, Dept. of Electronics and Communication,
BMS College of Engineering, Bengaluru, India

Abstract- *The Linux scheduler plays a critical role in optimizing resource allocation and task scheduling within the operating system. This paper presents a novel approach to enhance the Linux scheduler using machine learning (ML) techniques.*
The proposed ML engine leverages historical data on process behavior, system load, and resource utilization to make intelligent decisions about task scheduling and resource allocation. The engine utilizes a Gradient Boosting ML algorithm to capture complex patterns and relationships in the data.
Overall, this paper presents a pioneering approach to leverage ML techniques in the Linux scheduler. The proposed solution opens up new avenues for improving the efficiency and responsiveness of Linux-based systems in various domains, ranging from cloud computing to embedded devices.
Keywords— *Process Scheduling, Scheduler, Machine learning.*

R*eceived 04 July, 2023;* ***R****evised 12 July, 2023;* ***A****ccepted 14 July, 2023 © The author(s) 2023.*
P*ublished with open access at* ***www.questjournals.org***

I. INTRODUCTION

Process scheduling in an operating system (OS) scheduler is a crucial mechanism that determines the order in which processes are executed on a computer system. The primary objective of process scheduling is to efficiently allocate the limited system resources and provide fair execution to all processes. The OS scheduler employs various scheduling algorithms to achieve this goal. These algorithms take into consideration factors such as priority, burst time, arrival time, and other process attributes. Commonly used scheduling algorithms include First-Come-First-Serve (FCFS), Shortest Job Next (SJN), Round Robin (RR), Priority Scheduling, and Multilevel Queue Scheduling. Each algorithm has its advantages and trade-offs, balancing factors such as

turnaround time, response time, and resource utilization. The OS scheduler continuously monitors the state of processes, preempting and rescheduling them as necessary to ensure efficient and effective resource allocation, responsiveness, and overall system performance.

Process scheduling in an operating system (OS) scheduler is a crucial mechanism that determines the order and duration in which processes are executed on a CPU. One of the most widely acclaimed scheduling algorithms is the Completely Fair Scheduler (CFS), known for its superior fairness and efficiency. CFS achieves fairness by maintaining a balanced distribution of CPU time among all processes, aiming to provide each task with an equal share of the processor's resources. This is achieved by utilizing a red-black tree to keep track of the running processes and calculating the virtual runtime for each process. The CFS algorithm is considered the best choice for various reasons. Firstly, it ensures fairness by preventing any process from monopolizing the CPU, thereby preventing the degradation of system responsiveness. Secondly, it supports real-time process execution and prioritizes interactive tasks by dynamically adjusting the process priorities based on their interactive behavior. Additionally, CFS is highly scalable and efficient, making it well-suited for modern multi-core systems. Its ability to adapt to system loads and provide responsiveness to various workloads makes it an ideal choice for an OS scheduler, enabling optimal resource utilization and enhancing overall system performance.

II. LITERATURE SURVEY

[1] The paper introduces process scheduling techniques and discusses the memory layout of processes. It considers two types of executions: individual execution and parallel execution. Machine Learning (ML) algorithms are employed to take multiple attributes of processes into account and predict their Turn-around-time (TaT). These predictions are then utilized to assign appropriate time slices in the Linux scheduler. The experiments conducted in the paper demonstrate the successful reduction of TaT through this approach.

[2] This research highlights the potential of incorporating machine learning into operating system scheduling to optimize TaT. By considering both static and dynamic attributes and employing customized time slices, the study shows that machine learning-based predictive scheduling can lead to substantial improvements in process execution efficiency and resource utilization.

[3] The article introduces a model that utilizes Bayesian Decision Theory to optimize CPU scheduling. The model categorizes processes based on their properties and leverages a dataset of previously executed processes to enhance the selection of processes for scheduling. The primary objective is to improve scheduling in low-level language environments, specifically Assembly at the kernel level. By employing this approach, the model aims to enhance the efficiency and effectiveness of process selection, thereby improving overall CPU scheduling performance.

[4] The book aims to stay current, relevant, and adaptable to emerging needs in the field of operating systems education.

[5] CFS in Linux aims to provide fair process scheduling by maintaining a balanced tree structure and adjusting time slices based on process priorities represented by the nice value which ranges from -20 to +19. A lower nice value indicates higher priority, while a higher nice value indicates lower priority. The default value is 0.

[6] The study highlights the potential of combining machine learning-based burst time prediction with scheduling algorithms to optimize process scheduling in operating systems. The proposed ADRR algorithm offers a promising approach to minimize waiting time, process starvation, and context switches, leading to improved system performance and better resource utilization in scenarios with a large number of processes.

[7] The key takeaway from this study is that the proposed Median-Average Round Robin algorithm provides more effective results in terms of reducing average turnaround time and waiting time compared to traditional round-robin scheduling. This advancement highlights the potential for dynamic time quantum determination in improving the efficiency of CPU scheduling mechanisms.

[8] The study emphasizes the importance of reducing the overhead caused by process preemptions in schedulers and introduces a machine-learning-based approach to predict a better performing timeslice. This contributes to improved CPU cycle utilization and efficient system throughput.

[9] The paper highlights the potential of ML in resource scheduling for large-scale systems, showcasing its ability to improve efficiency, handle heterogeneity, and address scheduling challenges. The proposed ML-based solution offers a promising approach for achieving architectural optimization and enhancing resource management in computing environments.

[10] The research paper emphasizes the importance of leveraging Machine Learning to tackle the difficulties associated with rich media applications. It introduces a unique scheduling solution specifically designed for live omnidirectional video streaming in Unmanned Aerial Vehicle (UAV) environments. The proposed method outperforms existing scheduling approaches, leading to better outcomes in terms of user experiences and Quality of Service (QoS) in mobile applications. This innovative approach showcases the potential of Machine Learning

in optimizing scheduling decisions and achieving superior performance in challenging scenarios, thereby enhancing the overall quality and satisfaction of users.

III. PROBLEM STATEMENT

The Linux operating system uses a kernel scheduler to decide which processes to run and when to run them. The kernel scheduler is responsible for allocating CPU time to the different processes that are running on the system.

The Linux scheduler is a complex piece of software, but it can be summarized by the following four goals:

Maximize throughput: The scheduler should try to keep all of the CPUs busy, so that the system can run as fast as possible.
Minimize wait time: The scheduler should try to minimize the amount of time that processes have to wait before they can be scheduled for execution.
Minimize response time: The scheduler should try to minimize the amount of time that it takes for a process to start running after it is first created.
Fairness: The scheduler should try to give all of the processes a fair share of CPU time, regardless of their priority.
The Linux scheduler uses a variety of algorithms to achieve these goals. The default scheduler is the Completely Fair Scheduler (CFS), which is a multi-queue scheduler that gives each process a fair share of CPU time. The CFS scheduler also takes into account the priority of the processes, so that real-time processes are given higher priority than normal processes.

The Linux scheduler is a highly configurable piece of software. There are a number of parameters that can be tuned to optimize the scheduler for different workloads. For example, the scheduler can be configured to favor foreground processes over background processes, or to give more CPU time to processes that are interactive.

The Linux scheduler is a critical component of the Linux operating system. It is responsible for ensuring that the system can run as efficiently as possible, while also providing a fair and predictable environment for all of the processes that are running on the system.

In Linux, there are two main types of processes: real-time processes and normal processes.

Real-time processes are those that require guaranteed execution within a specified time period. These processes are often used for time-sensitive applications, such as controlling machinery or responding to network events. Real-time processes are given priority over normal processes, so they are more likely to be scheduled for execution.
Normal processes are those that do not require guaranteed execution within a specified time period. These processes are often used for less time-sensitive applications, such as word processing or web browsing. Normal processes are given lower priority than real-time processes, so they may not be scheduled for execution as often.

Real-time processes have the following characteristics:

- Deterministic Behavior: Real-time processes are executed with deterministic behavior, meaning their execution time and response time can be predicted and guaranteed.

- Fixed or Dynamic Priorities: Real-time processes are typically assigned fixed priorities or priorities that can be dynamically adjusted during runtime. Higher-priority real-time processes preempt lower-priority ones, ensuring that critical tasks are executed in a timely manner.

- Time Constraints: Real-time processes often specify timing constraints, such as deadlines or maximum execution times, to ensure their timely completion.

- Preemption: Real-time processes can preempt lower-priority processes if necessary to meet their timing requirements. This allows critical tasks to take immediate control of system resources.

Real-time processes are scheduled using specialized real-time scheduling algorithms and policies, such as SCHED_FIFO (First-In, First-Out) or SCHED_DEADLINE. These algorithms guarantee that real-time processes receive the necessary CPU time and resources to meet their deadlines, at the expense of potentially lower-priority processes.

Normal processes have the following characteristics:

- Variable Execution Time: Normal processes can have varying execution times, and their response time depends on the system load and the priority assigned to them.
- Priority-Based Scheduling: Time-shared processes are typically scheduled using priority-based algorithms, such as the Completely Fair Scheduler (CFS) in Linux. The scheduler assigns priorities dynamically based on factors like process behavior, CPU utilization, and interactivity to achieve fair resource allocation.
- Preemption: Normal processes can be preempted by higher-priority processes if necessary. This allows processes with higher priority, such as real-time processes or interactive processes, to gain immediate access to CPU resources when needed.

Normal processes often perform general-purpose tasks and are not subject to strict timing constraints. They can include system daemons, background processes, and various system tasks.

The load on the normal process scheduler in the Linux operating system poses significant challenges in terms of efficient resource allocation and system performance. As the number of normal processes increases, the scheduler's workload intensifies, resulting in increased overhead and potential degradation of overall system responsiveness. The scheduler's ability to fairly distribute CPU time among normal processes becomes strained, leading to suboptimal resource utilization and potential performance bottlenecks. In order to address this issue, there is a pressing need to explore solutions that can alleviate the load on the normal process scheduler, enhance its efficiency, and ensure optimal utilization of system resources for improved overall system performance.

IV. METHODOLOGY

Problem addressed

The high load on the normal process scheduler in Linux OS is a major issue that can have a significant impact on the user experience. The scheduler is responsible for deciding which processes to run and when to run them, and if it is overloaded, it can lead to delays in the execution of normal processes, as well as increased CPU usage. This can make the system seem slow or unresponsive, and it can also lead to overheating and battery drain.

In some cases, the high load on the normal process scheduler can also impact the performance of real-time processes, which can lead to failures or errors.

One way to address the high load on the normal process scheduler is to use a support engine. A support engine is a software application that can help to optimize the performance of the scheduler by identifying and resolving issues that are causing it to overload. This can be done by monitoring the scheduler's performance, identifying processes that are consuming excessive CPU time, and taking steps to optimize those processes.

By utilizing machine learning techniques, the following research aims to come up with a machine learning model which can act as a support engine that can make better predictions based on historical performance data and process characteristics. This approach helps reduce the load on the scheduler by minimizing the number of unnecessary context switches and preemptive actions. Ultimately, the goal is to optimize the scheduling decisions, improve system performance, and ensure efficient utilization of CPU resources.

Problem solution

To reduce the load on the scheduler, we have come up with an AI based support engine which can accurately classify the processes and thus reduce the load on the scheduler.

A supportive AI engine can play a crucial role in reducing the load on the scheduler for normal processes in a Linux operating system. The scheduler's primary responsibility is to allocate CPU time fairly among all processes, ensuring efficient resource utilization. However, as the number of processes increases, the scheduler may face challenges in managing and optimizing the scheduling decisions, potentially leading to increased overhead and decreased system performance.

By integrating a supportive AI engine, the system can offload certain tasks from the scheduler, thereby reducing its workload. The AI engine can intelligently analyze the behavior, resource requirements, and performance characteristics of normal processes over time. It can gather data on process execution patterns, resource utilization, and inter-process dependencies to build a comprehensive understanding of the workload.

Based on this analysis, the AI engine can make informed recommendations or decisions related to process scheduling. It can classify the processes into multiple classes, here three classes, and thereby reduce the scheduler overhead.

By collaborating with the scheduler, a supportive AI engine can alleviate its burden and enhance the efficiency of normal process scheduling. This integration allows the scheduler to focus on core responsibilities, such as handling real-time processes, while leveraging the AI engine's intelligent insights and adaptive capabilities to streamline the management of normal processes. Ultimately, this collaborative approach leads to better resource utilization, reduced scheduling overhead, and improved system responsiveness for normal process execution.

We have developed an ML (Machine Learning) model aimed at classifying processes into three distinct categories based on their previous resource utilization, priority values, nice values, and other relevant attributes. This classification framework leverages historical data and machine learning techniques to intelligently categorize processes and facilitate more efficient scheduling decisions.

The model takes into account various features extracted from the processes, such as CPU usage, memory consumption, I/O operations, execution times, and priorities assigned to them. It also considers attributes like the nice value, which represents the priority adjustment for a process in relation to other processes. By combining these features, the ML model can identify patterns and correlations that distinguish processes with similar resource requirements and behaviors.

Through a training phase, the ML model learns from labeled historical process data, capturing the relationships between process attributes and their associated categories. It uses algorithms such as decision trees, random forests, or neural networks to build a predictive model that generalizes from the training data and can classify unseen processes accurately.

Once trained, the model can be integrated into the process scheduling infrastructure. It analyzes process attributes, compares them with the learned patterns, and assigns processes to one of the predefined categories. These categories can represent different resource allocation strategies or scheduling policies based on the observed characteristics of the processes.

The benefits of this ML-based classification approach are manifold.

- Firstly, it enables a more nuanced understanding of the workload by grouping processes with similar characteristics together. This information can guide the scheduler in making informed decisions regarding resource allocation and scheduling priorities.
- Secondly, it reduces the burden on the scheduler by offloading the process categorization task to the ML model. This allows the scheduler to focus on other critical tasks, enhancing its efficiency.
- Lastly, by leveraging historical data, the model can adapt and improve over time, optimizing its classification accuracy and aligning with changes in process behavior.

In summary, our ML model offers a classification framework that categorizes processes into distinct groups based on their resource utilization, priority values, nice values, and other relevant attributes. By integrating this model into the process scheduling system, we aim to improve resource allocation, enhance the efficiency of the normal process scheduler, and ultimately optimize system performance.

V. PROPOSED ML MODEL FOR PROCESS PREDICTION

Data Collection and Preprocessing

The process of data collection and preprocessing is crucial for training an effective machine learning (ML) model for process prediction. In this section, we describe the steps involved in collecting and preprocessing the data.

Data collection involves gathering relevant information about the processes and their characteristics. This includes data such as process arrival time, elapsed time, CPU utilization, memory requirements, and process status. Depending on the availability and accessibility of data, it can be obtained from system logs, monitoring tools, or simulated environments.

Some of the process attributes that we have considered are: CPU utilization, memory utilization, process priority, nice value, and status of the process(example: sleeping, running, zombie etc).

Once the data is collected, preprocessing steps were applied to ensure its quality and suitability for ML model training. This involves tasks such as data cleaning, handling missing values, removing outliers, and normalizing and scaling the data to a consistent range. Preprocessing techniques also include feature extraction, where relevant features are extracted from raw data to represent the processes effectively.

Feature Engineering and Selection

Feature engineering involves transforming the raw data into a set of informative features that capture the characteristics of the processes. This includes creating new features based on domain knowledge or extracting statistical measures from existing features. Feature engineering helps in capturing important patterns and relationships within the data.

Feature selection is the process of identifying the most relevant and informative features for the ML model. It helps reduce dimensionality, eliminate redundant or irrelevant features, and improve the model's efficiency and accuracy. Techniques such as correlation analysis, feature importance ranking, and model-based feature selection are employed to identify the subset of features that have the most significant impact on process prediction. In our case, we have dismissed columns such as process ID, process name which do not contribute to the scheduling methodology.

ML Model Selection and Design

Various ML algorithms, such as decision trees, random forests, support vector machines, or neural networks, can be considered for process prediction. The selection of the ML model depends on factors such as the complexity of the problem, the nature of the data, and the desired interpretability or accuracy of the predictions.
There are a variety of ML models which can be used to address our problem statement. We have trained and tested our dataset on various types of models namely:

- Classification models which include Decision trees, K-nearest neighbors, Random forests, Gradient boosting.
- Regression model which includes Logistic Regression
- Clustering model which includes K-Means clustering

The design of the ML model involves defining the architecture, specifying the layers, activation functions, and optimization algorithms. This includes determining the appropriate input and output layers for the process prediction task. The model design should be tailored to handle the specific characteristics of the process data and leverage the insights gained from feature engineering.

Model Training and Evaluation

Model training and evaluation are essential steps in developing a robust and accurate ML model. In this section, we discuss the training and evaluation strategies employed in our research.

During the training phase, the ML model is presented with the preprocessed data, and the model parameters are adjusted to minimize the prediction errors. Training involves iteratively feeding the data to the model, updating the model's parameters, and fine-tuning the model's performance.
Evaluation of the trained model is performed using appropriate metrics such as accuracy, precision, recall, and mean squared error, depending on the nature of the process prediction task. The model is evaluated on separate test data to assess its generalization capabilities and performance on unseen examples.

Input Features for Process Representation

The selection of appropriate input features is crucial for accurately representing the process and enabling effective predictions. In this section, we discuss the input features used in our ML model for process representation. These features encompass both static and dynamic characteristics of the processes.

Some of the static features include the process priority, arrival time, memory requirements, and estimated execution time. These features provide information about the process characteristics that remain constant throughout its execution. Dynamic features, on the other hand, capture the runtime behavior of the process and include metrics such as CPU utilization, I/O wait time, and previous execution history. We have selected the static features of the process to schedule them, since normal processes usually come with static attributes and have no need to vary those attributes dynamically.

ML Model Architecture and Selection

The ML model architecture plays a crucial role in the accuracy and effectiveness of process prediction. In this section, we discuss the ML model architecture and the selection process employed in our research. Various ML algorithms, such as decision trees, neural networks, and support vector machines, can be considered for process prediction.
We carefully evaluated and compared different ML models based on their ability to handle the complexity of the process scheduling problem, interpretability, and prediction accuracy. After a thorough analysis, we selected the

Gradient Boosting model due to its ability to capture complex relationships within the data, handle both numerical and categorical features, and provide accurate predictions.

Training and Optimization Strategies
Training and optimization strategies are vital components in the development of an effective ML model for process prediction. In this section, we outline the training and optimization strategies employed in our research.

The training process involves feeding the ML model with labeled data, consisting of input features and corresponding process outcomes. The testing phase of an ML model is where the model is evaluated to see how well it performs on new data. This is done by feeding the model a set of unlabeled data, and then seeing how well the model can classify the data. The unlabeled data is data that has not been classified, so the model is not given any hints about how to classify it.

The testing phase is used to measure the generalization ability of the model. Generalization ability is the ability of the model to make accurate predictions on new data that it has not seen before. If the model is able to make accurate predictions on the testing data, then it is said to have good generalization ability.

VI. RESULTS

Dataset Description
The dataset consists of a collection of process instances, each characterized by various features such as arrival time, execution time, CPU utilization, and memory requirements.It also considers attributes like the nice value, which represents the priority adjustment for a process in relation to other processes. The dataset is representative of real-world scenarios and covers a diverse range of process behaviors and workload patterns.

To ensure the reliability and quality of the dataset, data collection was performed using reliable monitoring tools in a controlled environment. Special attention was given to include a sufficient number of process instances with varying characteristics to capture the diversity of workload scenarios.
The process data was collected from various systems which had the Linux operating system installed. We have also included data from systems which had varying system elements such as processor cores, system memory and so on.

Evaluation Metrics
To assess the performance of the ML model for process prediction, appropriate evaluation metrics are employed. In this section, we describe the evaluation metrics used to measure the effectiveness of the model.
Evaluation metrics include accuracy, precision, recall, and F1-score, which provide insights into the model's ability to correctly predict the next process in the scheduler. These metrics are calculated by comparing the predicted process with the actual process at each time step.

Accuracy measures assess the model's ability to predict process characteristics accurately, such as execution time or CPU utilization. These measures quantify the level of agreement between the predicted values and the actual values obtained during runtime. The higher the accuracy, the more reliable and effective the ML model is in predicting the next process in the scheduler.

Experimental Results:
We have trained and tested our dataset on various machine learning models described below:
Classification models:

1. Decision Trees: Decision trees are a type of supervised machine learning algorithm that can be used for both classification and regression tasks. They work by recursively partitioning the data into smaller and smaller subsets until each subset is classified or predicted.
2. K-nearest neighbors (KNN): It is a simple, non-parametric machine learning algorithm that can be used for both classification and regression tasks. It works by finding the k most similar points in the training dataset to a new point, and then using the labels of those points to predict the label of the new point.
3. Random Forest is an ensemble learning algorithm that combines multiple decision trees to make accurate predictions by aggregating their results.

4. Gradient Boosting is an ensemble learning technique that combines multiple weak learners (typically decision trees) to create a strong predictive model. It works by iteratively fitting new models to the residuals of the previous models, gradually reducing the prediction errors.

Regression models:

1. Logistic regression: Logistic regression is a statistical algorithm used for binary classification, which predicts the probability of an instance belonging to a particular class.It models the relationship between the dependent variable and one or more independent variables using a logistic function, also known as the sigmoid function.

Clustering models:

1. K-Means clustering is an unsupervised machine learning algorithm used for grouping similar data points into distinct clusters based on their similarity or distance from each other.

VII. DISCUSSION

Analysis of Prediction Accuracy

As per our research Gradient Boosting model had an accuracy score of 99% when trained on 30% of the input dataset.

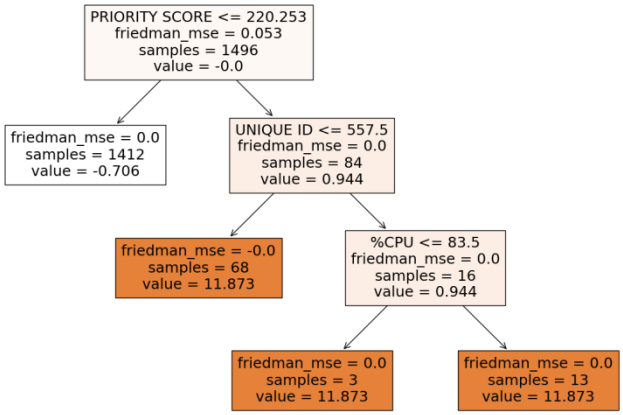

Figure-1

Figure-1 shows the graphical representation of the algorithm on the dataset.The key idea behind Gradient Boosting is that each new model is trained to correct the errors or residuals made by the previous models, leading to a sequence of models that collectively form a strong predictive ensemble. The process is guided by the gradient of the loss function, which determines the direction and magnitude of the updates at each boosting step.

Overall, Gradient Boosting is a powerful algorithm for building highly accurate predictive models by combining multiple weak models in an iterative manner.

Evaluation Metrics of Gradient Boosting model:

ACCURACY	0.997
PRECISION	0.989
RECALL	0.993
F1 SCORE	0.991

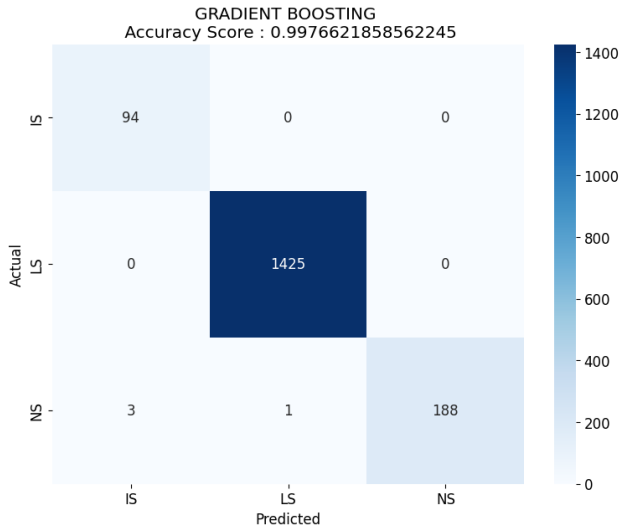

Figure-2

Figure-2 shows the confusion matrix,giving insights into the model's ability to classify tasks.

Insights into Process Behavior and Scheduling Efficiency

The experimental results provide valuable insights into the behavior of processes and the impact of the ML model on scheduling efficiency. By analyzing the prediction accuracy, waiting time reduction, and resource allocation optimization achieved by the ML model, we gain a deeper understanding of the relationship between process characteristics, scheduling decisions, and system performance.

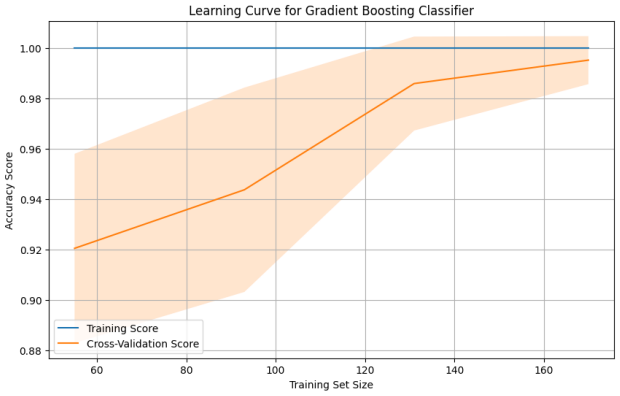

Figure-3

Figure-3 shows the learning curve of Gradient Boosting Algorithm,wider spread suggests high variance indicating the model to be sensitive to the training set.
The training score and cross validation score tend to converge,meaning the model is learning from data and achieving stable performance.
The findings shed light on the strengths and limitations of the ML model, highlighting areas for further improvement and optimization.

VIII. CONCLUSION

The table below shows the accuracy scores of all the models trained on the dataset

MODELS	ACCURACY SCORES
GRADIENT BOOSTING	0.99
RANDOM FOREST	0.96

DECISION TREE	0.99
KNN CLASSIFICATION	0.89
LOGISTIC REGRESSION	0.94

These findings highlight the effectiveness and potential benefits of incorporating ML techniques into the process scheduling domain.
The ML model demonstrates promising results in accurately classifying a process as either: IMMEDIATELY SCHEDULE, NEXT SCHEDULE or LATELY SCHEDULE, leading to improved scheduling efficiency and reduced waiting times. The model shows robustness and generalizability.

Future Research Directions

- Real-time Monitoring and Adaptation: Instead of relying solely on past resource utilization, develop mechanisms to monitor the current resource utilization of a process in real-time. This would enable the ML model to dynamically adjust predictions based on the changing resource demands of the process. Implementing feedback loops between the model and the resource allocation system can help optimize resource allocation on-the-fly.

- Advanced Time-Series Analysis: Explore advanced time-series analysis techniques to extract more meaningful patterns and trends from the historical resource utilization data. This could involve applying methods such as ARIMA (Autoregressive Integrated Moving Average), LSTM (Long Short-Term Memory), or other deep learning architectures designed for time-series data. These approaches may capture more complex relationships and improve the model's predictive capabilities.

By addressing these research directions, further advancements can be made in the field of process scheduling, leading to more efficient and intelligent scheduling algorithms that optimize system performance and resource utilization.

ACKNOWLEDGMENT

We would like to thank Dr. K N Madhusudhan, Assistant Professor in BMSCE, for giving us the opportunity to work on this topic. We would also like to thank our college (B.M.S. College of Engineering) for providing us the resources to work on the project.

REFERENCES

[1]. N. N. Jain, S. K. R and S. Akram, "Improvising process scheduling using machine learning," 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2018, pp. 1379-1382, doi: 10.1109/RTEICT42901.2018.9012330.
[2]. A. Negi and P. K. Kumar, "Applying Machine Learning Techniques to Improve Linux Process Scheduling," TENCON 2005 - 2005 IEEE Region 10 Conference, Melbourne, VIC, Australia, 2005, pp. 1-6, doi: 10.1109/TENCON.2005.300837.
[3]. Aslam, N., Sarwar, N. and Batool, A., 2016. Designing a model for improving CPU scheduling by using machine learning. International Journal of Computer Science and Information Security, 14(10), p.201.
[4]. Silberschatz, Abraham, P.B. Galvin, and G. Gagne, "Operating system concepts", Vol. 8. Wiley, 2013.
[5]. CFS scheduler - The linux kernel archives, Linux Kernel Source Code (2.6.23/2.6.24): sched-design-CFS.txt by Ingo Molnar. [Online]. Available: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
[6]. M. M. A. Moni, M. Niloy, A. H. Chowdhury, F. J. Khan, M. F. -U. -A. Juboraj and A. Chakrabarty, "Comparative Analysis of Process Scheduling Algorithm using AI models," 2022 25th International Conference on Computer and Information Technology (ICCIT), Cox's Bazar, Bangladesh, 2022, pp. 587-592, doi: 10.1109/ICCIT57492.2022.10055395.
[7]. Sakshi, Chetan Sharma, Shamneesh Sharma, Sandeep Kautish, Shami A. M. Alsallami, E.M. Khalil, Ali Wagdy Mohamed,"A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time", Alexandria Engineering Journal,Volume 61, Issue 12, 2022, Pages 10527-10538, ISSN 1110-0168, https://doi.org/10.1016/j.aej.2022.04.006.
[8]. Ojha, P., Thota, S.R., Vani, M. and Tahilianni, M.P., 2015. Learning Scheduler Parameters for Adaptive Preemption. CS & IT-CSCP.
[9]. Yang, R., Ouyang, X., Chen, Y., Townend, P. and Xu, J., 2018, March. Intelligent resource scheduling at scale: a machine learning perspective. In 2018 IEEE symposium on service-oriented system engineering (SOSE) (pp. 132-141). IEEE.
[10]. Comşa, I.S., Muntean, G.M. and Trestian, R., 2020. An innovative machine-learning-based scheduling solution for improving live UHD video streaming quality in highly dynamic network environments. IEEE Transactions on Broadcasting, 67(1), pp.212-224.