Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification

A Project Report

Submitted by

MINNU ANTONY

MAC21CS035

to

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering

Mar Athanasius College of Engineering (Autonomous)

Kothamangalam

April 2025

Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification

A Project Report

Submitted by

MINNU ANTONY

MAC21CS035

to

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering

Mar Athanasius College of Engineering (Autonomous)

Kothamangalam

April 2025

DECLARATION

I undersigned hereby declare that the project report **Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification** , submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, is a bonafide work done by me under supervision of **Prof. Eldo P Elias** and **Prof. Rotney Roy Meckamalil**. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Minnu Antony

Date :

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MAR ATHANASIUS COLLEGE OF ENGINEERING

(AUTONOMOUS)

KOTHAMANGALAM

CERTIFICATE

This is to certify that the report entitled **"Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification"** submitted by **Ms. Minnu Antony (MAC21CS035)** towards partial fulfillment of the requirement for the award of Degree of Bachelor of Technology in Computer Science and Engineering from APJ Abdul Kalam Technological University for April 2025 is a bonafide record of the project carried out by her under our supervision and guidance.

Prof. Eldo P Elias	**Prof. Rotney Roy Meckamalil**	**Prof. Joby George**
Project Guide	Project Coordinator	Head of Department

Internal Examiner(s) **External Examiner(s)**

Date : Dept.Seal

ACKNOWLEDGEMENT

I express our sincere gratitude and thanks to Dr. Bos Mathew, Principal and Prof. Joby George, Head of the Department for providing the necessary facilities and their encouragement and support.

I owe special thanks to the project guide Prof. Eldo P Elias and project co-ordinator Prof. Rotney Roy Meckamalil for their corrections, suggestions and sincere efforts to co-ordinate the project under a tight schedule.

I express my sincere thanks to staff members in the Department of Computer Science and Engineering who have taken sincere efforts in guiding and correcting me in conducting this project.

Finally, I would like to acknowledge the heartfelt efforts, comments, criticisms, co-operation and tremendous support given by my dear friends during the preparation of the project and also during the presentation without which this work would have been all the more difficult to accomplish.

ABSTRACT

Optimizing system performance while ensuring efficient resource utilization is critical in modern computing. This project, "Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification," presents a machine learning-driven framework that dynamically tunes system parameters based on workload characteristics. By integrating workload simulation, data collection, and predictive modeling, the system continuously adjusts kernel parameters to maximize performance and throughput. XGBoost is used for workload classification and throughput prediction, enabling precise, real-time system adjustments. A kernel tuning mechanism performs Linux system parameter optimization to adapt to varying workloads, ensuring responsiveness and stability. The adaptive mechanism of this system ensures that system parameters are optimized to support varied workloads and resource needs. Results show that dynamic tuning beats static settings in terms of workload throughput and system responsiveness. This adaptive and scalable solution is applicable to enterprise data centers, cloud computing systems, and home computing systems, leading to more efficient use of energy in computing environments with high computational dependability.

Contents

LIST OF FIGURES

ABBREVIATIONS

ML Machine Learning

DL Deep Learning

SLA Service Level Agreement

LHS Latin Hypercube Sampling

XGBoost Extreme Gradient Boosting

I/O Input/Output

RAM Random Access Memory

LLC Last-level Cache

CHAPTER 1

INTRODUCTION

Optimizing system performance with minimal energy use has been a main challenge in contemporary computing environments. Energy-efficient computing is particularly important in data centers, cloud computing environments, and high-performance computing systems, where workloads change dynamically, and resource usage has to be managed tightly. Poor system configurations can result in wasteful power usage, shortened hardware life, and higher operating expenses. Thus, the development of intelligent, adaptive systems that optimize energy consumption without compromising computational efficiency is critical to sustainable computing.

This project aims at creating a parameter tuning framework that uses machine learning algorithms to improve system efficiency[1]. The basic concept is to adaptively tune system parameters in response to real-time workload patterns. The proposed solution combines workload simulation, energy measurement collection, and prediction modeling to implement an automated, smart tuning framework. Through generating workload samples and measuring energy usage, our framework constructs models for classifying workloads, predicting their computational intensity, and providing optimal system recommendations.

Advanced machine learning methods applied to workload detection and system tuning are one of the most prominent features of this project. Models like XGBoost is employed to accurately classify workloads and forecast system behavior with varying parameter configurations. In addition, there is an automated kernel parameter tuning mechanism, which dynamically adjusts Linux system parameters to optimize performance and energy efficiency. Through these methods, we seek to close the gap between legacy static config-

urations and contemporary adaptive computing environments.

A set of experiments is rigorously conducted to mimic multiple workloads and assess their energy efficiency under various configurations. The report outlines the sequential execution process, ranging from workload generation to model training and parameter optimization. Additionally, the effect of the machine learning-based method is examined by comparing system performance and throughput across different configurations. These analyses provide valuable insights into the effectiveness of the framework and its applicability in real-world scenarios.

By integrating workload detection, predictive modeling, and adaptive tuning mechanisms, this project contributes to the broader field of resource-efficient computing. The results demonstrate how machine learning can be effectively utilized to enhance system performance [1] while optimizing the workload throughput. This framework has the potential to be implemented across diverse environments, from enterprise-level data centers to home computing platforms, offering an intelligent and scalable solution for power-aware computing. The findings highlight the growing role of artificial intelligence in system optimization and open pathways for future advancements in energy-conscious computing technologies.

CHAPTER 2

BACKGROUND

In this chapter, the necessary technologies and frameworks required to build the project, "Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification", are discussed.

2.1 Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence that deals with creating algorithms that allow computers to learn from data and make decisions or predictions without direct programming. There exists multiple ML models which are trained on datasets containing structured or unstructured data, and these models can be improved over time. It is typically classified as supervised, unsupervised, and reinforcement learning. Supervised learning is mostly employed in this project to classify workloads based on past data.

Machine Learning finds its uses across a wide range of fields, such as natural language processing, computer vision, fraud detection, and autonomous systems.The project utilizes machine learning to categorize workloads effectively, enabling correct resource allocation within computing environments.

2.2 Deep Learning

Deep Learning (DL) is a category of ML concerned with neural networks with more than one layer. Deep architectures enable models to learn hierarchical representations of raw data automatically. Deep learning methods are best applied when one is working with large amounts of complex data like images, time series, and unstructured text.

Deep learning techniques can improve workload classification using architectures like Convolutional Neural Networks (CNNs) for feature extraction and Recurrent Neural Networks (RNNs) or Transformers for sequential analysis of workloads.

2.3 Machine Learning Frameworks

ML frameworks provide essential tools, libraries, and APIs for implementing, training, and deploying machine learning models. In this project, we use industry-standard ML frameworks that facilitate workload classification and model optimization. The major frameworks relevant to our work include:

2.3.1 TensorFlow

TensorFlow is an open-source ML platform that was created by Google. It has a large collection of tools for developing and deploying ML models in varied environments such as CPUs, GPUs, and TPUs. TensorFlow has high-level APIs such as Keras, which make it easy to implement deep learning models. TensorFlow Extended (TFX) can also be used for the deployment and monitoring of ML models in production environments.

2.3.2 PyTorch

PyTorch is an open-source deep learning platform created by Facebook AI Research. It has dynamic computation graphs, which improve model debugging and experimentation. PyTorch is heavily utilized for academic research and industry implementation because it supports easy use, robust GPU acceleration, and support for deep learning models such as CNNs, RNNs, and Transformers.

2.3.3 Scikit-Learn

Scikit-Learn is one of the widely used ML libraries in Python, offering a comprehensive collection of tools for data preprocessing, feature extraction, and model training. Scikit-Learn offers a wide range of ML algorithms including Support Vector Machines (SVMs), Decision Trees, and ensemble techniques like Random Forests and Gradient Boosting.

Due to its efficiency and simplicity, Scikit-Learn is utilized in our project for baseline model development and comparative performance analysis.

2.3.4 XGBoost

XGBoost (Extreme Gradient Boosting) is a highly optimized gradient boosting framework for high-performance machine learning applications. It is greatly effective in performing structured data classification and regression problems. In our project, XGBoost is utilized in order to enhance the efficiency and accuracy of workload classification by efficiently managing feature importance and missing values.

2.4 Machine Learning Algorithms for Workload Classification

Several ML algorithms play a crucial role in workload classification and resource optimization. These include:

2.4.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) is a strong supervised learning method employed for classification and regression. SVM works on finding the best hyperplane which optimally discriminates between various classes in high-dimensional space. The major advantage of SVM is its capability to work with linear and nonlinear classification through the utilization of kernel functions like polynomial, radial basis function (RBF), and sigmoid kernels.

In workload categorization, SVM is employed to classify workloads into categories according to their patterns of resource usage. By projecting workload attributes into a higher-dimensional space, SVM can successfully differentiate among various types of workloads and enable better resource allocation and optimization.

2.4.2 Decision Trees and Random Forest

Decision Trees is a basic but effective classification technique that divides data according to feature values to produce a tree-shaped model of decisions. Every node is a feature test,

and every branch is an outcome. Splitting goes on until a stopping condition is reached, i.e., a minimum number of samples in a leaf node.

Random Forests extend Decision Trees by aggregating several trees into an ensemble. This ensemble method avoids overfitting and enhances generalization by averaging the predictions of several decision trees. Each tree is trained on a random subset of the data, adding variability that enhances the overall model's strength.

Random Forests can be used to classify the workloads by their attributes to minimize misclassifications and achieve good generalizability of the model to unforeseen workload patterns. Random Forests are ideal when handling structured data and perform very well with workload classification.

2.4.3 Gradient Boosting Algorithms

Gradient Boosting is an ensemble learning method that creates models sequentially, where each model improves upon the previous one by correcting its mistakes. It is based on decision trees as the base learners and optimizes them with a gradient descent algorithm.XGBoost, LightGBM, and CatBoost are well-known implementations of gradient boosting, each with training speed and prediction accuracy improvements. These models manage missing values well, accommodate parallel processing, and offer feature importance information. In our project, Gradient Boosting algorithms are employed to classify workloads with high accuracy. Through iteratively improving workload classification predictions, Gradient Boosting assists in enhancing resource allocation efficiency and system performance.

CHAPTER 3

RELATED WORKS

Optimizing cloud server performance requires efficient workload classification and dynamic parameter tuning. Over the years, researchers have explored various approaches, ranging from conventional workload management strategies to advanced machine learning techniques. This section highlights key developments in workload classification, prediction, parameter tuning, and multi-system workload optimization.

3.1 Conventional Workload Management Techniques

Traditionally, workload management relied on static configurations and rule-based heuristics to optimize system parameters. A widely used technique, Dynamic Voltage and Frequency Scaling (DVFS), adjusts CPU frequency based on system utilization to reduce power consumption [2], particularly for CPU-intensive tasks. While DVFS is effective in specific scenarios, it falls short when dealing with diverse workloads that demand optimized memory, disk, and network performance.

Another conventional approach involves rule-based resource scheduling, where predefined thresholds determine resource allocation. However, these static rules often fail to adapt to real-time workload fluctuations, leading to inefficient resource usage and potential performance bottlenecks.

3.2 Machine Learning-Based Workload Classification

As workloads in cloud environments become more complex, machine learning techniques have been employed to classify them based on performance metrics. Earlier approaches

used models like k-Nearest Neighbors (KNN) and Support Vector Machines (SVM) to categorize workloads by analyzing system counters and usage patterns.

More advanced techniques, such as XGBoost and Neural Networks, have demonstrated higher classification accuracy, leveraging large datasets to capture subtle workload variations. Some studies also apply Principal Component Analysis (PCA) to reduce feature dimensions and improve classification efficiency. However, these methods often require large labeled datasets, making real-world implementation challenging due to the cost and effort involved in data labeling.

3.3 Workload Prediction Techniques

Predicting workload trends in cloud environments is essential for proactive resource management. Various methods have been explored to forecast resource demand, helping data centers allocate resources efficiently while avoiding Service Level Agreement (SLA) violations[3].

1. **Statistical Methods:** Traditional approaches like Auto-Regressive Integrated Moving Average (ARIMA) [4] and Hidden Markov Models (HMM) analyze past trends to predict future workloads. These techniques perform well in stable environments but struggle with highly variable workloads.

2. **Machine Learning Models:** Techniques like Support Vector Regression (SVR) and Bayesian Ridge Regression (BRR) [5] improve prediction accuracy by identifying patterns in resource usage. However, they require careful tuning to avoid overfitting or inaccurate predictions in dynamic environments.

3. **Deep Learning Approaches:** Advanced models such as Long Short-Term Memory (LSTM) networks [6] and Deep Belief Networks (DBN) can capture complex workload behaviors, making them ideal for cloud workload prediction. Despite their

accuracy, these models demand significant computational power and large training datasets, making real-time implementation a challenge.

Recent studies also explore clustering-based workload prediction, where workloads with similar behavioral patterns are grouped, and specialized models are trained for each group. This technique significantly improves prediction accuracy compared to a single model applied to all workloads.

3.4 Multi-Attribute Workload Prediction for System Optimization

Predicting workload behavior is essential for maintaining system efficiency and optimizing resource allocation. Traditional methods often focus on a single aspect, such as CPU usage, but real-world workloads involve multiple factors, including memory, disk, and network activity. A more comprehensive approach considers these attributes together, providing a clearer picture of system demands.

Support Vector Regression (SVR) has been explored for modeling complex, non-linear workload patterns [4], offering improved accuracy through optimized kernel functions and parameter tuning techniques. By analyzing multiple resource attributes, predictive adjustments can be made dynamically, enhancing system responsiveness and preventing performance degradation. SVR leverages statistical learning techniques and cross-validation to refine predictions, reducing errors and improving adaptability in dynamic environments.

Compared to conventional approaches, multi-attribute workload prediction enables better adaptation to varying conditions [4], allowing for more effective tuning of system parameters to balance resource utilization and throughput. Proactive adjustments based on workload trends help mitigate bottlenecks, leading to a more stable and efficient system performance over time.

3.5 Parameter Tuning for Cloud Workloads

Dynamically adjusting system parameters plays a crucial role in optimizing cloud performance. Several techniques have been explored to fine-tune kernel settings and hardware configurations for better energy efficiency and throughput.

Metaheuristic algorithms [7], such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), search for optimal parameter combinations by simulating natural selection and swarm intelligence. Reinforcement Learning (RL) has gained attention for its ability to continuously adapt system parameters based on observed performance metrics [8]. However, RL-based approaches can introduce high computational overhead, making them unsuitable for real-time tuning. A promising strategy combines DVFS with kernel parameter tuning, where CPU frequency adjustments are paired with optimizations in memory management, I/O scheduling, and network parameters. This hybrid approach helps balance performance and energy consumption, particularly for workloads with varying resource demands.

3.6 Multi-System Workload Optimization

Optimizing workloads across multiple servers is another growing area of research. Instead of tuning parameters on a single machine, multi-system coordination enables distributed resource optimization.

Federated Learning-Based Tuning [5] allows multiple cloud systems to share workload patterns and collectively improve tuning efficiency while preserving data privacy. Workload Migration Strategies dynamically shift workloads between servers to balance resource usage and prevent performance degradation in heavily loaded machines. These approaches enhance scalability and adaptability in cloud environments, but efficient communication frameworks are required to minimize data transfer overhead and ensure seamless coordination between systems.

3.7 Linux Performance Optimization for Enhanced System Efficiency

Linux, an open-source operating system, is widely used for its flexibility and efficiency across various computing environments. Optimizing its performance requires monitoring key metrics such as CPU usage, memory utilization, disk I/O, and network throughput [9].

Performance Monitoring Tools like *top*, *htop*, *iostat*, and *netstat* help analyze system performance, while techniques such as process management, kernel tuning, CPU frequency scaling, and multithreading enhance efficiency.

Memory and Disk Optimization involve adjusting swappiness, managing caches, leveraging RAID configurations, selecting efficient file systems, and implementing caching strategies to enhance performance.

Network Performance Tuning, including TCP/IP parameter adjustments and traffic shaping, optimizes system responsiveness.

Advanced Optimization Techniques like system-wide tuning with *tuned*, service management with *systemd*, and optimizations for virtualization and containerization ensure sustained performance.

Continuous monitoring and iterative tuning are essential to maintaining an optimized Linux system, allowing it to operate at peak efficiency across various workloads.

3.8 Intelligent Task Scheduling and System Optimization

Intelligent task scheduling plays a crucial role in optimizing system performance by dynamically allocating resources based on workload characteristics. Traditional scheduling methods such as First Come, First Serve (FCFS) and Shortest Job First (SJF) often lead to inefficiencies due to their inability to adapt to workload variations. Recent advancements leverage machine learning models, including decision trees and XGBoost, to predict workload behavior and optimize scheduling decisions, reducing execution latency and improving system responsiveness.

Dynamic system tuning further enhances performance by adjusting CPU scheduling,

memory management, and disk I/O settings in real time. Studies have shown that reinforcement learning-based techniques can continuously optimize system parameters, leading to improved throughput and resource utilization. Additionally, in cloud computing environments, energy-aware scheduling algorithms like the Energy Management Algorithm (EMA) [9] help balance computational loads efficiently across virtual machines.

The integration of workload classification with real-time system optimization enables computing environments to dynamically adjust to workload fluctuations, ensuring optimal performance. As research in adaptive scheduling continues to evolve, intelligent resource management frameworks are becoming increasingly relevant to cloud platforms, high-performance computing systems, and enterprise data centers.

CHAPTER 4

DESIGN AND IMPLEMENTATION

The core objective of this project is to design and implement a workload classification and Linux kernel tuning framework that dynamically adjusts system parameters based on workload demands. The system integrates machine learning-based classification and kernel parameter optimization.This section provides a detailed breakdown of the system architecture, workload classification model, optimization framework, and the implementation details, including code structure and execution flow.

4.1 System Architecture

4.1.1 Overview

The system is designed to optimize resource utilization by classifying workloads and adjusting system parameters accordingly. It consists of multiple interconnected components that facilitate workload generation, machine learning-based classification, and dynamic optimization of system resources.

The core idea behind the system is to improve efficiency in handling computational tasks by dynamically adjusting system configurations based on workload characteristics. Traditional static resource allocation often leads to inefficiencies, either over-provisioning or under-utilizing system resources [8]. By employing machine learning techniques, the system can classify workloads in real-time and make data-driven adjustments to system parameters, ensuring optimal resource usage and enhanced performance.

The system integrates machine learning models with Linux system management tools to continuously monitor and classify workloads. It can be applied in cloud environments, high-performance computing (HPC) clusters, and enterprise data centers where resource

optimization plays a crucial role in reducing costs and improving execution speed.

4.1.2 System Components

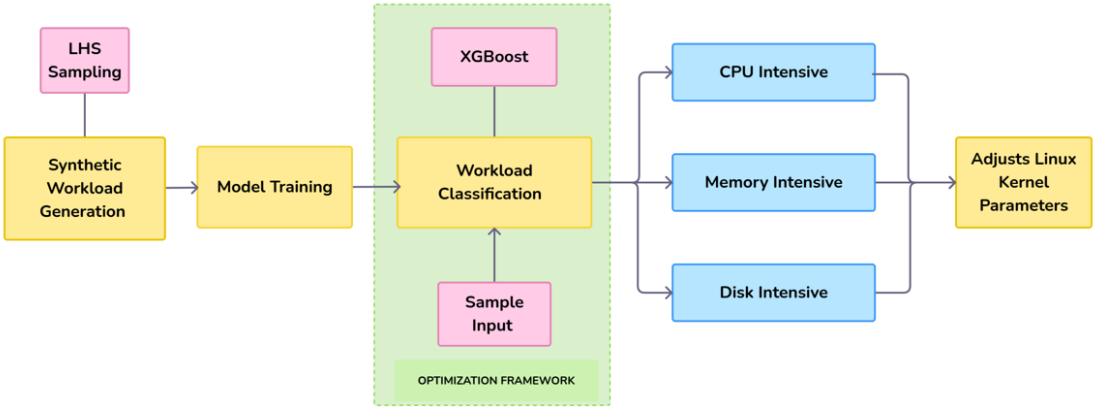

Figure 4.1: Basic Architecture of the proposed framework

The architecture (Fig 4.1) of the system is structured into the following key components:

1. Synthetic Workload Generation: The system generates diverse workloads to simulate real-world computational tasks.Latin Hypercube Sampling (LHS) is used to create a representative set of synthetic workloads with varied characteristics.

2. Model Training: Feature extraction is performed using system performance counters.The extracted features are used to train a machine learning model.The primary classification algorithm employed is XGBoost, known for its efficiency in handling structured data and its robustness in decision-making.

3. Workload Classification: The trained model classifies incoming workloads based on resource consumption patterns. Classification categories include CPU-intensive, Memory-intensive, and Disk-intensive workloads.New workloads are fed into the model, and their characteristics are analyzed to assign the appropriate category.

4. Optimization Framework: Based on workload classification, the system determines optimal system parameters.The classified workloads are mapped to specific Linux kernel parameter adjustments to enhance resource efficiency.

5. Linux Kernel Parameter Tuning: The system modifies key kernel parameters dynamically to optimize performance for different workload types. Adjustments are made to CPU scheduling, memory management, and disk I/O handling to ensure efficient execution of classified workloads.

6. Integration with Linux Systems: The system operates within a Linux environment, leveraging system-level monitoring tools to extract real-time performance metrics.Custom scripts and automation tools are used to adjust kernel settings dynamically based on classification results.

This architecture ensures an efficient and adaptive system capable of optimizing resource utilization through intelligent workload classification and targeted parameter tuning.

4.2 Synthetic Workload Generation

Efficient workload generation and accurate classification are the foundation of our system's ability to dynamically tune Linux kernel parameters. To achieve this, we designed a methodology that ensures the generated workloads are diverse, representative, and systematically distributed across different resource usage patterns. Once these workloads are executed, we collect system performance metrics to train a machine-learning model capable of recognizing workload characteristics. The classification model then serves as the decision-making engine for adjusting kernel parameters.

4.2.1 Latin Hypercube Sampling (LHS) for Synthetic Workload Generation

One of the critical challenges in workload generation is ensuring that the dataset used for training is diverse enough to cover a wide range of real-world scenarios. Instead of relying on simple random sampling, which often results in redundant or unbalanced data points, we employ Latin Hypercube Sampling (LHS) to systematically distribute workload parameters across the input space.

LHS ensures that each parameter is evenly represented without excessive clustering of values, leading to a more comprehensive workload dataset. For instance, when defining

synthetic workloads, several factors must be considered, such as CPU utilization, memory consumption, and disk activity. Using LHS, we generate a set of workload configurations where each configuration is unique and spans different levels of intensity across these factors. This approach helps in capturing the full spectrum of system behaviors, making the subsequent classification model more robust and generalizable. These measurements serve as the foundation for training the classification model. By ensuring that workloads are well-distributed across different resource consumption patterns, the classification model can learn to distinguish between different workload types with high accuracy.

	cpu_load	memory_load	disk_load
2	0.314193617912444	0.0741549060228795	0.77870112260307
3	0.551332007669033	0.654300647175943	0.635411384961331
4	0.384700989304053	0.456669748915846	0.80387684561276
5	0.357978559744718	0.53761762395706	0.818745839341249
6	0.582005295095432	0.602226124753567	0.166817003571201
7	0.862900814092192	0.0306963727630295	0.779519813469512
8	0.0969504290146955	0.606110622135968	0.474134633099537
9	0.384366753961397	0.588664646089007	0.752714387271157
10	0.597041240061204	0.2685469272274	0.784801166667922
11	0.339253165184136	0.377379502113605	0.323388644420236
12	0.899919826071904	0.237748052394903	0.930046157494394
13	0.851886968810844	0.251797849606734	0.592067466617822
14	0.0524403273920912	0.739425718023877	0.430214231981926
15	0.09781558551021	0.966400669696468	0.268062904528793
16	0.653646340431008	0.908259862493751	0.640244311337392
17	0.797710466841541	0.195032196025741	0.449904855331237
18	0.253125637651811	0.971963732257774	0.422589049542487
19	0.553718348059193	0.662926641604312	0.00591472539054183
20	0.340283029827644	0.13402589241387	0.511421300760052

Figure 4.2: Workload configurations generated by LHS

4.3 Model Training and Classification

With a well-structured dataset obtained through LHS-based workload generation, the next step is to develop a classification model that can automatically identify the nature of a given workload based on system performance data. To accomplish this, we use a combination of feature selection and machine learning techniques, particularly XGBoost (Extreme Gradient Boosting), a highly efficient and accurate classification algorithm.

4.3.1 Feature Selection and Performance Counters

The effectiveness of the classification model depends heavily on the choice of input features. Since workloads can stress different subsystems of a machine, it is essential to select

performance counters that provide meaningful insights into system behavior. For CPU-intensive workloads, relevant metrics include CPU cycles, cache misses, and instructions per cycle (IPC), as these directly indicate how much computational power a process is consuming. Memory-intensive workloads, on the other hand, are best characterized by RAM usage, page faults, and swap activity, since they involve frequent memory access and management. Finally, disk-intensive workloads can be identified through features such as disk throughput, I/O wait time, and read/write speeds.

These system performance counters are collected in real time [10] as workloads are executed, forming a labeled dataset that is used to train the classification model. The goal is to ensure that each workload type has distinct statistical characteristics that allow the model to differentiate between them accurately.

4.3.2 XGBoost for Workload Classification

Given the complexity of the relationships between system performance counters and workload behavior, a powerful and flexible machine-learning model is required for classification. We chose XGBoost, a decision-tree-based ensemble algorithm, due to its ability to capture non-linear dependencies in the data while remaining computationally efficient. XGBoost has been widely used for high-dimensional classification problems, making it a suitable choice for our workload classification task.

The model training process begins with preprocessing the collected performance data. This involves normalizing the values to account for scale differences between different system counters and removing any redundant or highly correlated features that do not contribute significantly to classification accuracy. Once the data is cleaned, the model is trained on a portion of the dataset, with hyperparameters such as tree depth and learning rate optimized using cross-validation.The model is tested on unseen workloads to ensure that it can generalize well to new data. Through this process, XGBoost learns to associate specific patterns in system performance metrics with different types of workloads, enabling real-time classification during system operation.

4.3.3 Classification of Workload Types

Once trained, the classification model is capable of analyzing system performance data and determining whether the current workload is CPU-intensive, memory-intensive, or disk-intensive. This classification step is crucial, as it forms the basis for adjusting Linux kernel parameters dynamically to optimize resource allocation.

A CPU-intensive workload, for example, involves heavy computation and frequent execution of instructions, leading to high processor utilization. To accommodate such workloads, kernel parameters related to CPU scheduling and frequency scaling can be adjusted to prioritize performance. Memory-intensive workloads, on the other hand, place significant pressure on the system's RAM and paging mechanisms. In this case, modifying parameters related to memory management, such as swappiness and dirty writeback settings, can help maintain system stability and efficiency. Finally, disk-intensive workloads involve high read/write operations, necessitating adjustments to disk caching policies and I/O schedulers to reduce latency and improve throughput.

The classification model continuously analyzes workload behavior and dynamically assigns it to one of these categories. By doing so, it provides an essential input to the optimization framework, allowing the system to proactively adjust kernel settings in response to changing workload conditions

4.4 Integration with Kernel Parameter Tuning

Once a workload is classified into one of the predefined categories—CPU-intensive, memory-intensive, or disk-intensive—the system dynamically adjusts the relevant **Linux kernel parameters** to optimize resource allocation. The objective is to fine-tune system behavior in response to workload demands, thereby improving throughput, reducing latency, and ensuring efficient resource utilization.

WORKLOAD TYPES	PARAMETERS
CPU - Intensive	vm.swappiness
Memory - Intensive	vm.swappiness vm.dirty_background_ratio vm.dirty_ratio vm.min_free_kbytes
Disk - Intensive	vm.dirty_expire_centisecs vm.dirty_writeback_centisecs

Figure 4.3: Significant Linux Kernel Parameters

4.4.1 CPU-Intensive Workload Adjustments

CPU-intensive workloads require efficient processor scheduling and cache management to minimize execution time. To optimize system behavior for such workloads, we adjust the following parameter:

- **vm.swappiness**: Controls how aggressively the kernel swaps memory pages to disk. Lowering vm.swappiness reduces swapping, ensuring that active processes remain in RAM for faster execution. A typical adjustment for CPU-intensive workloads is setting vm.swappiness = 10 to prioritize in-memory execution.

4.4.2 Memory-Intensive Workload Adjustments

Memory-bound workloads require careful tuning of virtual memory management settings to prevent excessive paging and ensure efficient memory allocation. The following kernel parameters are adjusted:

- **vm.swappiness**: Similar to CPU workloads, but in memory-intensive scenarios, this value is set based on workload needs. A moderate value (e.g., vm.swappiness = 30) helps balance RAM and swap usage.

- **vm.dirty_background_ratio**: Determines the percentage of system memory filled with dirty pages before background writeback starts. Lowering this value (e.g., vm.dirty_background_ratio = 5) ensures faster and more frequent writes to disk, reducing memory congestion.

- **vm.dirty_ratio**: Defines the percentage of system memory that can be occupied by dirty pages before writes are forced. Setting vm.dirty_ratio = 20 helps prevent excessive accumulation of dirty pages, improving writeback efficiency.

- **vm.min_free_kbytes**: Specifies the minimum amount of free memory that the kernel should maintain. Increasing this value prevents memory exhaustion and ensures smoother performance under high memory usage.

4.4.3 Disk-Intensive Workload Adjustments

For workloads with high disk I/O activity, the system optimizes disk writeback policies to improve I/O performance and prevent bottlenecks. The following parameters are adjusted:

- **vm.dirty_expire_centisecs**: Controls how long dirty pages remain in memory before being written to disk. Reducing this value (e.g., vm.dirty_expire_centisecs = 1500) ensures more frequent writes, reducing sudden spikes in disk activity.

- **vm.dirty_writeback_centisecs**: Determines how often the kernel's background process flushes dirty pages to disk. Setting this to a lower value (e.g., vm.dirty_writeback_centise = 500) results in a smoother and more continuous writeback process.

By implementing this kernel tuning approach, we ensure that system performance remains optimized across varying workloads without manual intervention. This adaptive tuning mechanism significantly enhances resource utilization, system responsiveness, and overall efficiency.

4.5 Implementation Strategy

4.5.1 Software Tools and Technologies Used

The implementation of the system relies on a combination of programming languages, machine learning frameworks, and system optimization tools. Python serves as the primary language for workload classification and optimization, leveraging libraries such as NumPy, Pandas, and Scikit-learn for data processing and model training. XGBoost is used for workload classification, while system-level optimizations are applied through Linux kernel parameters.

4.5.2 Code Structure and Execution Flow

The project follows a modular design with clearly defined components for workload data collection, classification, and system tuning. The core components include:

- **Data Preprocessing Module:** Collects and processes system performance metrics.
- **Classification Module:** Implements the trained XGBoost model for workload classification.
- **Optimization Module:** Maps workload categories to appropriate kernel adjustments.

The execution follows a structured pipeline:

1. Collect system performance metrics from workload execution.
2. Classify the workload type based on real-time data.
3. Apply corresponding kernel optimizations to enhance resource utilization.
4. Monitor and validate the system's performance after tuning.

4.5.3 Integration with Linux Systems

The system is designed to work seamlessly with Linux-based environments. Kernel tuning is achieved through dynamic modification of system parameters using tools like `sysctl` and `psutil`. The classification and optimization modules are executed as user-space applications that interact with the Linux kernel to adjust system parameters based on the classified workload type.

Through this structured implementation, the system effectively improves resource utilization, reducing overhead and enhancing overall performance for diverse workload types.

CHAPTER 5

EVALUATION AND ANALYSIS

5.0.1 Workload Classification Performance

The performance of the workload classification model was evaluated based on accuracy. The trained XGBoost model achieved a classification accuracy of approximately 90%, as observed over multiple epochs. The validation accuracy trend indicates stable learning, with minimal overfitting. The model's ability to generalize across different workload types ensures that system optimizations are applied correctly based on workload characteristics.

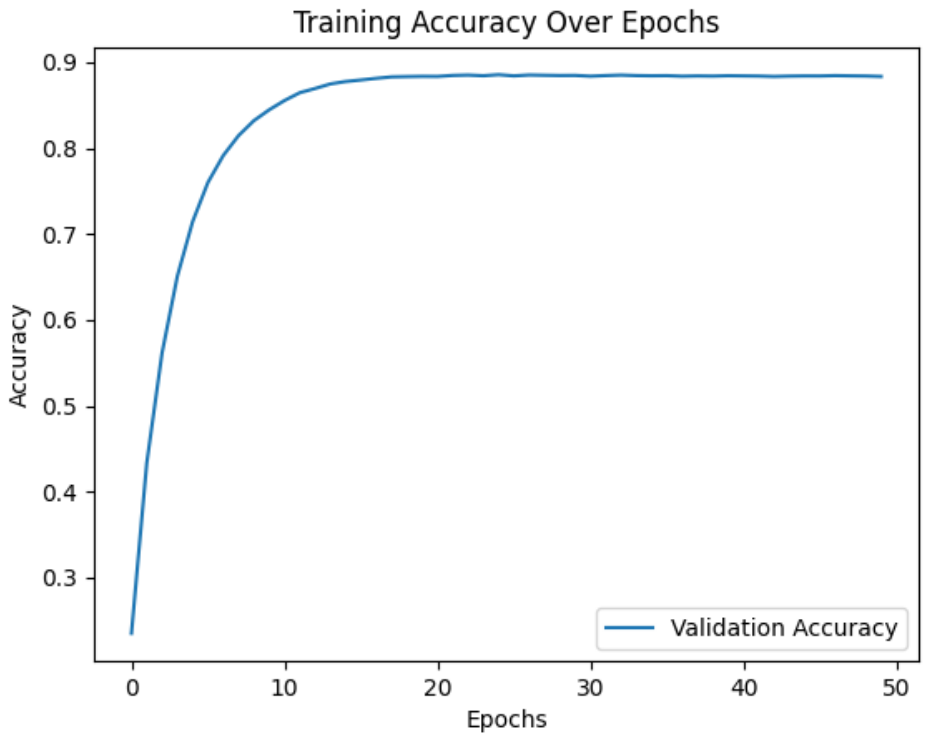

Figure 5.1: Training Accuracy Over Epochs

5.0.2 Kernel Parameter Adjustments

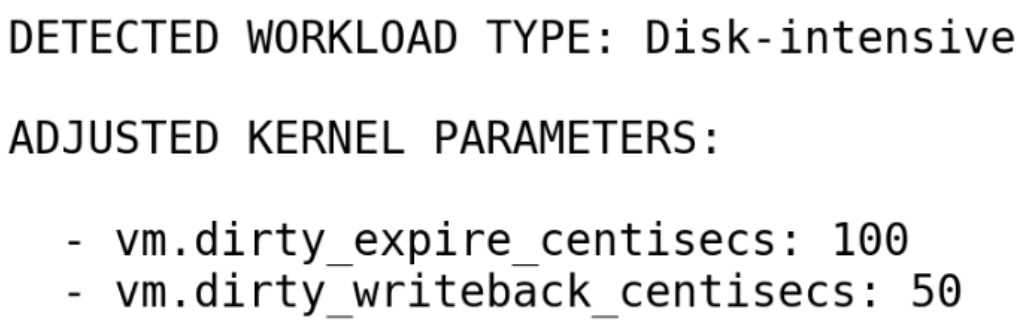

Figure 5.2: Sample Output

Upon classification, workload-specific system optimizations were applied:

- **CPU-Intensive Workloads:** Adjusted `vm.swappiness` to balance memory swapping, ensuring that CPU-bound processes are not slowed down by unnecessary memory paging.

- **Memory-Intensive Workloads:** Modified `vm.dirty_background_ratio`, `vm.dirty_ratio`, and `vm.min_free_kbytes` to optimize memory utilization and prevent excessive swapping, which could degrade performance.

- **Disk-Intensive Workloads:** Tuned `vm.dirty_expire_centisecs` and `vm.dirty_writeback_centisecs` to improve disk write efficiency, reducing latency in disk-heavy processes.

5.0.3 Performance Improvement Analysis

To assess the impact of tuning, throughput was estimated before and after applying optimizations. Across all workload types, the system demonstrated measurable enhancements in efficiency.

For CPU-intensive workloads, reducing excessive swapping allowed processes to execute with lower latency, leading to a more responsive system. Memory-intensive workloads benefited from improved memory allocation policies, preventing excessive page faults and reducing cache misses. Disk-intensive workloads saw an improvement in I/O responsiveness by ensuring timely writes and reducing disk congestion.

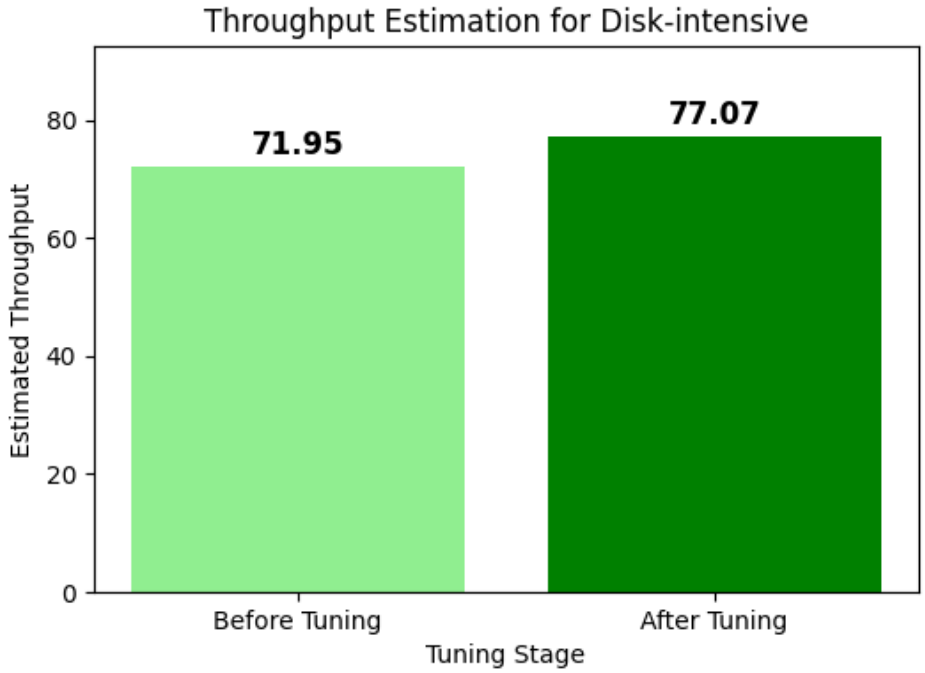

Figure 5.3: Throughput Before and After Tuning

The analysis also considered performance counter metrics such as CPU cycles, last-level cache (LLC) misses, and interrupts, which provided insights into how the system responded to different workload classifications. The observed improvements validate the effectiveness of adaptive tuning in optimizing resource utilization and maintaining workload efficiency.

5.0.4 Summary of Findings

- The classification model effectively distinguished between workload types with high accuracy, allowing precise tuning of system parameters.
- Workload-specific kernel optimizations resulted in performance improvements across

CPU, memory, and disk-intensive workloads.

- The applied optimizations ensured better resource utilization, reduced processing bottlenecks, and improved overall system efficiency without introducing significant overhead.
- Performance counter analysis confirmed that tuning resulted in tangible improvements in system responsiveness and resource allocation.

This evaluation demonstrates the effectiveness of machine learning-based workload classification and adaptive system tuning in optimizing resource utilization while maintaining stable and efficient system performance.

CHAPTER 6

FUTURE SCOPE

The future of workload classification and kernel parameter tuning presents several opportunities for enhancement, particularly in automation, adaptability, and broader system integration. One significant area for advancement is enabling real-time adaptation to workload variations. Currently, the system classifies workloads based on predefined execution patterns, but incorporating dynamic detection and automated tuning would allow for more responsive and efficient resource management. Leveraging machine learning techniques such as reinforcement learning or Bayesian optimization could enable the system to adjust kernel parameters intelligently, minimizing the need for manual intervention and improving overall performance.

Another important direction is the expansion of performance metrics used in workload classification. While the current approach relies on system counters, integrating additional factors such as power consumption, thermal efficiency, and I/O latency could provide a more comprehensive view of system behavior. This would allow for finer-grained optimizations, balancing performance with energy efficiency and system stability. Additionally, ensuring compatibility with diverse hardware architectures, including ARM and RISC-V, would make the system more versatile and applicable across various computing environments.

Implementing this system in cloud environments presents a promising opportunity for future development. Cloud platforms host highly dynamic and heterogeneous workloads, making intelligent resource management crucial for optimizing performance and cost efficiency. By adapting workload classification and kernel tuning strategies for cloud-based environments, the system could help improve virtual machine performance, reduce latency, and enhance resource utilization. Edge computing environments could also benefit

from this approach, ensuring efficient workload handling closer to the data source while maintaining optimal system performance. Future research could also explore hybrid approaches combining LSTM-based forecasting models like UtilML [11] with reinforcement learning for dynamic system optimization

Furthermore, enhancing user customizability would make the system more adaptable to different computing needs. Allowing users to define workload-specific performance goals—such as prioritizing latency, energy efficiency, or throughput—would enable more tailored tuning strategies [12]. With these advancements, the project can evolve into an intelligent, fully automated system capable of optimizing workload execution across a wide range of computing platforms, from local systems to large-scale cloud environments.

CHAPTER 7

CONCLUSION

Machine learning-driven workload classification and system optimization provide a structured approach to enhancing resource utilization and improving system performance. By employing XGBoost for workload classification and dynamically tuning Linux kernel parameters based on workload characteristics, measurable performance gains were achieved across CPU, memory, and disk-intensive tasks. The adaptive approach ensures that system resources are allocated efficiently, minimizing bottlenecks and improving overall throughput. The evaluation of the classification model demonstrated high accuracy in identifying workload types, enabling precise tuning of kernel parameters. For CPU-intensive workloads, reducing unnecessary memory swapping led to lower execution latencies and improved responsiveness. Memory-intensive workloads benefited from optimized memory allocation policies, preventing excessive paging and reducing cache misses. Disk-intensive workloads saw reduced I/O congestion and improved write efficiency, leading to enhanced system stability and throughput.

Performance counter metrics, including CPU cycles, cache misses, and interrupts, provided deeper insights into the system's behavior before and after tuning, validating the effectiveness of adaptive optimizations. Additionally, real-time tuning mechanisms could further enhance responsiveness by continuously adjusting parameters based on evolving workload patterns. The results underscore the impact of machine learning in intelligent resource management, demonstrating its potential to create more adaptive and self-optimizing computing environments. By automating workload classification and system tuning, this approach lays the groundwork for scalable, efficient, and high-performance computing systems capable of adapting to diverse workloads in real time.

References

[1] Jiechao Liang, Weiwei Lin, Yifan Xu, et al. Energy-aware parameter tuning for mixed workloads in cloud server. *Cluster Computing*, 27:4805–4821, July 2024.

[2] Weiwei Lin, Xiaoxuan Luo, ChunKi Li, Jiechao Liang, Guokai Wu, and Keqin Li. An energy-efficient tuning method for cloud servers combining dvfs and parameter optimization. *IEEE Transactions on Cloud Computing*, 11(4):3643–3655, 2023.

[3] Lamees M. Al Qassem, Thanos Stouraitis, Ernesto Damiani, and Ibrahim Abe M. Elfadel. Proactive random-forest autoscaler for microservice resource allocation. *IEEE Access*, 11:2570–2585, 2023.

[4] Labeb Abdullah, Huixi Li, Shamsan Al-Jamali, Abdulrahman Al-Badwi, and Chang Ruan. Predicting multi-attribute host resource utilization using support vector regression technique. *IEEE Access*, 8:66048–66067, 2020.

[5] Jiechao Gao, Haoyu Wang, and Haiying Shen. Machine learning based workload prediction in cloud computing. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2020.

[6] Nevlin T Noble, Yadu P Dev, and Christina Terese Joseph. Machine learning based techniques for workload prediction in serverless environments. In *2023 International Conference on Electrical, Electronics, Communication and Computers (ELEXCOM)*, pages 1–6, 2023.

[7] Ivo Pereira, Ana Madureira, Eliana Costa e Silva, and Ajith Abraham. A hybrid metaheuristics parameter tuning approach for scheduling through racing and case-based reasoning. *Applied Sciences*, 11(8), 2021.

[8] Chandrakanth Lekkala. Ai-driven dynamic resource allocation in cloud computing:

Predictive models and real-time optimization. *J Artif Intell Mach Learn & Data Sci*, 2(2), February 6 2024. Available at SSRN.

[9] Adeel Ahmed, Muhammad Adnan, Saima Abdullah, Israr Ahmad, Nazik Alturki, and Leila Jamel. An efficient task scheduling for cloud computing platforms using energy management algorithm: A comparative analysis of workflow execution time. *IEEE Access*, 12:34208–34221, 2024.

[10] Houkun Zhu, Dominik Scheinert, Lauritz Thamsen, Kordian Gontarska, and Odej Kao. Magpie: Automatically tuning static parameters for distributed file systems using deep reinforcement learning. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, pages 150–159, 2022.

[11] Christian Bauer, Narges Mehran, Radu Prodan, and Dragi Kimovski. Machine learning based resource utilization prediction in the computing continuum. In *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 219–224, 2023.

[12] Linux Journal Staff. Elevate your linux experience: Effective performance optimization techniques for enhanced speed, 2024. Accessed: March 13, 2025.

[13] P. Nehra and Nishtha Kesswani. A workload prediction model for reducing service level agreement violations in cloud data centers. *Decision Analytics Journal*, 11:100463, 2024.

phase2_report.pdf

ORIGINALITY REPORT

4%	4%	2%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.coursehero.com Internet Source	4%