Optimizing System Resource Utilization Using Machine Learning-Based Workload Classification

Minnu Antony[1*], Negha Ajayaghosh[2*],
Varsha Harikumar[3*],
Eldo P Elias[4*],Rotney Roy Meckamalil[5*]
*Department of Computer Science and Engineering
Mar Athanasius College of Engineering (Autonomous), Kothamangalam, Kerala
[1]minnuantony03@gmail.com, [2]neghaajayaghoshofficial@gmail.com,
[3]varsha1708harikumar@gmail.com, [4]eldope@gmail.com [5]rotney.rotney@gmail.com

Abstract—**Optimizing system performance while ensuring efficient resource utilization is critical in modern computing. Here, a machine learning-driven framework is proposed to dynamically tune 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 utilized for workload classification and throughput prediction, enabling precise, real-time system adjustments. A kernel tuning mechanism optimizes Linux system parameters to adapt to varying workloads, ensuring responsiveness and stability. The adaptive nature of this system ensures that system parameters are continuously optimized to support diverse workloads and resource demands. Experimental results demonstrate that dynamic tuning outperforms static configurations in terms of workload throughput and system responsiveness. This adaptive and scalable solution is applicable to enterprise data centers, cloud computing environments, and personal computing systems, contributing to more efficient energy utilization and enhanced computational dependability.**

I. Introduction

Modern computing environments are characterized by a diverse range of workloads, each with distinct computational, memory, and storage requirements. High-performance computing (HPC), cloud platforms, and enterprise data centers process workloads that vary significantly in structure and resource demands, ranging from compute-intensive numerical simulations to data-driven analytics and large-scale storage operations. Efficiently managing system resources in such dynamic environments remains a fundamental challenge, as the performance of a system is highly dependent on the align- ment between workload characteristics and resource allocation strategies.

Traditional system optimization methods often rely on static configurations, where predefined parameter settings govern how computational resources such as CPU cycles, memory allocation, and disk I/O are managed. While effective for predictable workloads, these methods struggle to accommodate workload variability, leading to suboptimal performance when system demands shift. A configuration optimized for a compute-heavy application may lead to unnecessary memory paging in a data-intensive workload, while disk-intensive tasks may suffer from inefficient I/O scheduling if parameters remain unchanged. These inefficiencies contribute to increased execution latency, resource bottlenecks, and degraded system responsiveness, making static tuning an inadequate solution for modern computing workloads.

The ability to dynamically adapt system parameters in response to workload fluctuations [1] is critical for maintaining optimal performance. Recent advancements in workload-aware optimization leverage machine learning techniques to analyze system performance metrics and classify workload behavior in real time. By continuously monitoring key indicators such as CPU utilization, memory access patterns, disk throughput, and cache performance, intelligent optimization frameworks can proactively adjust system parameters to match workload characteristics. This adaptive approach mitigates the limitations of static configurations by ensuring that resource management strategies remain aligned with the evolving demands of the system.

One of the key advantages of workload-aware optimization is its ability to enhance computational efficiency across a wide range of applications [2]. Compute-intensive workloads benefit from optimized CPU scheduling and reduced context switching, memory-bound applications experience fewer page faults and improved allocation efficiency, and disk-intensive workloads achieve faster read/write speeds through dynamic I/O tuning. The ability to fine tune system behavior based on real-time workload classification enables more effective utilization of computational resources, reducing unnecessary overhead while maintaining high system throughput.

Experimental evaluations indicate that dynamically adjusting system parameters based on workload classification results in measurable improvements in performance [1] [2]. By leveraging data-driven optimization techniques, computing environments can become more responsive to workload variations, reducing processing delays and improving overall system efficiency. These findings suggest that workload-aware system tuning presents a promising direction for improving computational performance in diverse computing environments, enabling more adaptive and intelligent resource management strategies.

II. Related Works

Optimizing system resource utilization has been a significant area of research in computing, particularly as modern infrastructures demand high efficiency and adaptability. Traditional methods of resource management often relied on static configurations or predefined rules that dictated how system resources were allocated. While these approaches provided some level of optimization, they lacked the flexibility needed to respond to the constantly changing demands of real-world workloads. As computing environments became more dynamic—especially with the rise of cloud computing and large-scale data centers—there emerged a need for intelligent workload classification and adaptive tuning mechanisms.

Machine learning (ML) has recently transformed the way system optimization is approached. By leveraging data-driven decision-making, ML-based techniques can classify workloads, predict system behavior, and adjust parameters dynamically to achieve optimal performance. These advancements have led to more refined strategies for workload-aware resource allocation, reducing energy consumption and improving overall system efficiency. This section explores the evolution of workload management, ranging from traditional static configurations to advanced ML-driven optimization strategies.

A. Traditional Workload Management Techniques

Historically, workload management relied on predefined policies that controlled how system resources were allocated. These policies were often based on heuristic approaches that set static thresholds for CPU, memory, and disk usage. While effective in stable environments, they struggled to adapt to fluctuating workload patterns, leading to inefficiencies such as resource underutilization or over allocation.

One widely used traditional technique was Dynamic Voltage and Frequency Scaling (DVFS) [2], which dynamically adjusted the CPU's operating frequency and voltage based on workload demands. The goal of DVFS was to balance power consumption with performance, reducing energy usage when computational demand was low. However, DVFS had limitations—it primarily targeted CPU-bound tasks and did not account for the varying demands of memory-intensive or disk- heavy workloads. Moreover, DVFS was often reactive rather than proactive, meaning that it adjusted parameters only after detecting workload changes, sometimes leading to delays in performance improvements.

Apart from DVFS, rule-based scheduling algorithms were commonly employed to manage resource distribution [3]. These included static provisioning methods where resources were pre-allocated based on expected workload behavior. Such approaches lacked real-time adaptability and often resulted in inefficient resource usage. As workload diversity increased in cloud computing and HPC environments, researchers recognized the need for more dynamic and intelligent workload management strategies.

B. Machine Learning for Workload Classification

The rise of machine learning introduced a paradigm shift in workload classification, allowing systems to learn from historical data and make predictive adjustments. Traditional classification techniques such as Support Vector Machines (SVM), Decision Trees, and k-Nearest Neighbors (k-NN) [4] were among the early methods used to categorize workloads based on observed system metrics. These models were capable of identifying workload types by analyzing CPU utilization, memory access patterns, and disk I/O activities.

While these classical ML models provided a foundation for workload classification, they had limitations, particularly when dealing with high-dimensional or nonlinear data. En- semble learning methods, such as Extreme Gradient Boosting (XGBoost) [5] [6], emerged as a more effective alternative. XGBoost is an optimized gradient boosting algorithm known for its ability to handle large datasets efficiently while cap- turing complex relationships between workload characteristics and resource demands. By training on labeled workload data, XGBoost can classify workloads more accurately and predict how they will impact system performance.

Another advancement in workload classification involved the use of feature selection techniques to improve model efficiency. Instead of analyzing all available system metrics, researchers employed methods such as Principal Component Analysis (PCA) and recursive feature elimination to identify the most relevant workload characteristics. This helped reduce computational overhead while maintaining high classification accuracy.

C. Predictive Modeling for System Optimization

Predicting workload behavior is essential for proactive resource allocation [7], as it allows systems to anticipate demand and adjust parameters before performance bottlenecks occur. Early attempts at workload prediction relied on statistical models such as Auto-Regressive Integrated Moving Average (ARIMA) and Hidden Markov Models (HMM) [8]. These techniques analyzed historical workload data to identify trends and forecast future resource requirements. However, they were primarily designed for stable and periodic workloads, making them less effective in highly dynamic cloud environments.

To overcome these challenges, researchers explored deep learning models such as Long Short-Term Memory (LSTM) [9] [10] networks for workload prediction. LSTMs, a type of recurrent neural network (RNN), are particularly well-suited for time-series forecasting because they can retain memory of past states and capture long-term dependencies in data. By training on workload traces from cloud systems, LSTMs can predict CPU, memory, and disk usage patterns with high accuracy [9]. Another promising approach is reinforcement learning (RL)-based optimization, where an agent continuously learns to adjust system parameters based on real-time feedback [11]. RL techniques such as Deep Q-Networks (DQN) enable sys- tems to explore different tuning strategies and identify optimal configurations that maximize performance while minimizing energy consumption. While reinforcement learning holds sig- nificant potential for real-time system tuning, its computational requirements remain a challenge for large-scale deployment.

D. *Dynamic Kernel Parameter Tuning*

Tuning Linux kernel parameters dynamically has gained attention as an effective way to optimize system performance. Unlike static configurations, which rely on fixed settings for CPU scheduling, memory management, and disk I/O, dynamic tuning mechanisms adjust these parameters in real-time based on workload characteristics.

Several studies have explored metaheuristic algorithms [12], such as Genetic Algorithms (GA) and Particle Swarm Opti- mization (PSO), to optimize kernel parameters. These algo- rithms search for the best combination of settings by sim- ulating natural selection or swarm intelligence. For example, GA-based tuning strategies evolve optimal configurations over multiple iterations by selecting high-performing parameter sets and refining them through crossover and mutation.

Additionally, Bayesian optimization [13] has been used for fine-tuning kernel parameters by modeling system perfor- mance as a probabilistic function. This technique efficiently explores the parameter space and identifies configurations that minimize energy consumption while maintaining performance. Automated tuning frameworks, such as Tuned and AutoTuner, have also been developed to adjust kernel parameters dynam- ically based on predefined performance goals.

E. *Multi-System Workload Optimization*

In distributed computing environments, optimizing work- loads across multiple servers presents additional challenges. Unlike single-system tuning, where resource allocation is managed within a single machine, multi-system workload optimization involves balancing workloads across a network of servers [14].

One approach to tackling this problem is federated learning- based workload tuning [15], where multiple cloud servers collaboratively train a shared workload classification model while preserving data privacy. By sharing model updates rather than raw data, federated learning enables distributed optimization without exposing sensitive workload information. Another technique, workload migration strategies, involves dynamically shifting workloads between servers to balance resource utilization. For example, virtual machine (VM) mi- gration techniques allow workloads to be moved between physical hosts to avoid overloading a single machine. Con- tainer orchestration platforms, such as Kubernetes, [16] auto- mate workload placement across cloud environments, ensuring efficient resource distribution.

Although multi-system workload optimization offers im- proved scalability and fault tolerance, it introduces new com- plexities, such as network latency, synchronization overhead, and inter-server communication costs. These challenges must be carefully managed to ensure seamless workload distribu- tion.

F. *Advancements in Machine Learning for Workload Predic- tion*

Traditional workload prediction models, such as Auto- Regressive Integrated Moving Average (ARIMA) [17] and Vector AutoRegression (VAR), have been widely applied in time-series forecasting to estimate resource demands. These models work well for stationary data but struggle with non-linear and highly dynamic workloads commonly found in serverless and cloud computing environments. To address these limitations, deep learning models such as Long Short-Term Memory (LSTM) networks have been introduced, demonstrating superior accuracy in workload forecasting by capturing complex temporal dependencies. The use of reinforcement learning-based [17] task allocation in serverless computing has also been explored. One study proposed a Gray Wolf Optimization (GWO)-based reinforcement learning approach [18] for task scheduling, demonstrating improved execution efficiency and reduced resource wastage. Additionally, prediction-based autoscaling techniques have gained prominence, enabling cloud platforms to dynamically adjust computing resources based on forecasted workload demands.

III. Methodology

To optimize system resource utilization dynamically, the proposed framework integrates workload classification with adaptive kernel parameter tuning [1]. This methodology consists of four essential components: workload generation, performance data collection, machine learning-based workload classification, and dynamic system tuning. Each of these components plays a crucial role in ensuring that the system can intelligently adapt to different workload demands in real-time, enhancing overall efficiency while maintaining system stability.

A. *Workload Generation*

The first step in the proposed methodology involves creating a dataset that represents a diverse set of workload conditions. Workload behavior varies significantly based on the nature of computational tasks, making it essential to simulate different workload types to train a reliable machine learning model. In- stead of relying on existing datasets, which may not capture the full range of workload variations,*Latin Hypercube Sampling (LHS)* [19] to generate synthetic workloads systematically was employed.

LHS is a statistical sampling method that ensures a well- distributed selection of input parameters. Unlike random sam- pling, which may result in redundant or unbalanced workload distributions, LHS guarantees that each parameter (CPU usage, memory utilization, disk I/O activity) is evenly represented across different intensity levels. This allows to create a bal- anced dataset that accurately captures the behavior of various workload types.

In this study, workloads are categorized into three primary groups:

- **CPU-intensive workloads**: Tasks such as cryptographic operations, mathematical computations, and AI model inference, which place significant stress on the processor while requiring minimal disk or memory operations.
- **Memory-intensive workloads**: Large-scale database op- erations and scientific simulations, which demand fre- quent memory allocation and high cache utilization, lead- ing to increased RAM consumption and potential page swapping.
- **Disk-intensive workloads**: Processes that involve ex- tensive read and write operations, as seen in tasks like file indexing, video encoding, and log processing. These workloads put a significant load on the I/O subsystem and require careful tuning of disk scheduling policies.

To generate and execute these workloads, *benchmarking tools such as sysbench and fio* were used, which allowed to create controlled workload environments. During execution, system performance metrics are continuously recorded to form the dataset for training the workload classification model.

B. Performance Data Collection

Once workloads are generated and executed, it is essen- tial to monitor system behavior and capture real-time per- formance metrics. These metrics provide insights into how different workloads interact with system resources, allowing the machine learning model to identify patterns and classify workloads accurately.

To collect performance data, built-in *Linux monitoring tools*, including `vmstat`, `iostat`, `top`, and `perf` were utilized. These tools provide real-time insights into CPU utilization, memory consumption, disk activity, and hardware-level per- formance counters. The collected data includes a variety of performance indicators, such as CPU load percentage, memory allocation, cache misses, page faults, disk read/write speeds, and system interrupts.

For instance, when executing a CPU-intensive workload, it is expected to observe high CPU utilization, minimal disk activity, and a steady allocation of memory. In contrast, a memory-intensive workload would exhibit frequent page accesses, increased cache usage, and potential memory swap- ping, while a disk-intensive workload would generate high disk throughput with noticeable fluctuations in I/O wait times.

The collected performance data is stored in structured logs, which are later processed to train the machine learning model. These logs form a labeled dataset, where each recorded system state is associated with the corresponding workload type. Before training, the data undergoes preprocessing steps, including feature normalization to ensure that different performance metrics are on a comparable scale.

C. Machine Learning-Based Workload Classification

With a comprehensive dataset of system performance met- rics, the next step is to develop a machine learning model capable of classifying workloads in real-time [15]. The ability to accurately distinguish between CPU-intensive, memory- intensive, and disk-intensive workloads is crucial for imple- menting targeted system optimizations.

To achieve high classification accuracy,*Extreme Gradient Boosting (XGBoost)*, a decision-tree-based ensemble learning algorithm known for its speed and effectiveness in handling structured data was used. XGBoost has several advantages over traditional classification models, including its ability to capture complex, non-linear relationships between input features and its robustness against noisy data.

Before training the model, *feature selection* was performed to identify the most relevant system performance indicators that contribute to workload differentiation. The dataset is split into **80% training and 20% testing**, ensuring that the model is exposed to a variety of workload conditions during training. To enhance performance, *hyperparameter tuning* is conducted, adjusting parameters such as learning rate, tree depth, and regularization factors.

Once deployed, the trained model continuously analyzes real-time system performance data, classifying workloads dynamically as they are executed. This classification enables the system to make intelligent adjustments to kernel parameters, optimizing performance based on workload characteristics.

D. System Parameter Tuning

After a workload is classified, the system must respond by adjusting *Linux kernel parameters* to optimize resource allocation dynamically. Different workloads place varying demands on system resources, necessitating workload-specific optimizations to enhance throughput and reduce inefficiencies. For **CPU-intensive workloads**, the primary objective is to ensure that processor resources are utilized efficiently while minimizing unnecessary context switches. One of the key adjustments is reducing the swappiness parameter (vm.swappiness = 10), which prevents excessive swapping of active processes to disk, thereby ensuring that CPU-bound tasks remain in RAM. Additionally, CPU scheduling policies are optimized to prioritize computationally intensive tasks.

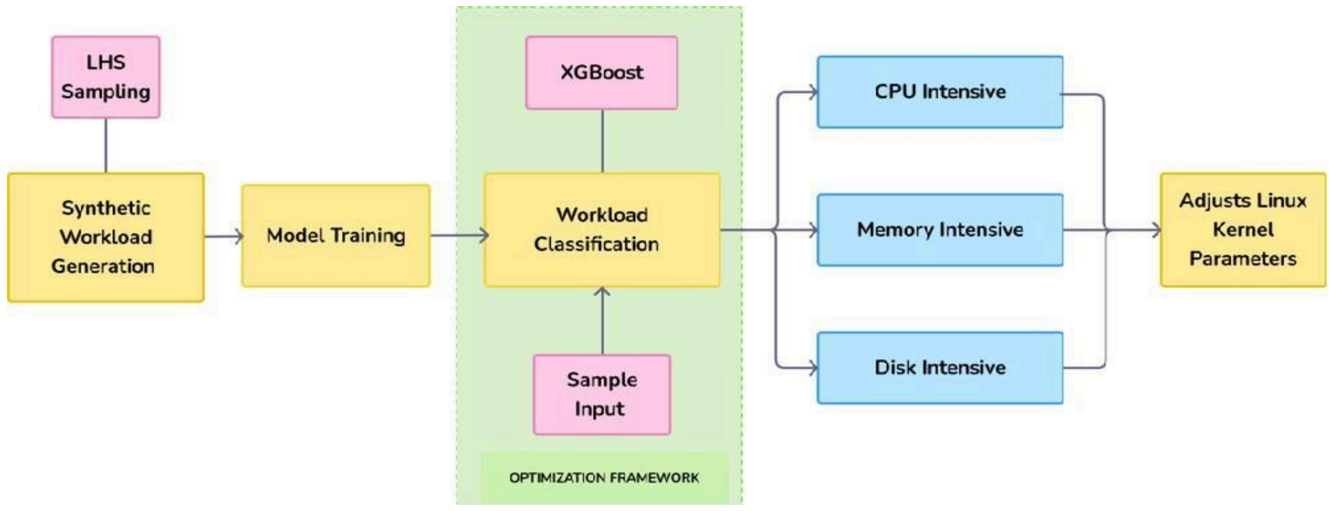
In the case of **memory-intensive workloads**, optimizing virtual memory management is crucial to prevent excessive paging and ensure efficient RAM utilization. The dirty ratio (vm.dirty_ratio = 20) is adjusted to control the percentage of system memory that can be occupied by dirty pages before they are written to disk. The *minimum free memory threshold* (vm.min_free _kbytes) is also increased to avoid memory ex- haustion, ensuring that critical system processes continue to function without interruptions.

For **disk-intensive workloads**, optimizing I/O operations is key to maintaining high throughput and minimizing latency. The *dirty expire centisecs* (vm.dirty_expire_centisecs = 1500) parameter is modified to ensure that dirty pages are flushed to disk more frequently, preventing sudden bursts of I/O activity that could degrade performance.

To implement these dynamic adjustments, *sysctl* and *psutil* commands were utilized, which allow kernel parameters to be modified in real-time without requiring a system reboot. A background daemon continuously monitors workload classifications and applies appropriate tuning adjustments as needed. This ensures that the system remains responsive and adapts to changing workload demands seamlessly. A similar approach has been proposed in [20], focusing on optimizing workflow execution time while balancing energy consumption in cloud environments. By integrating workload classification with adaptive kernel tuning, the outlined approach ensures that system resources are allocated efficiently, leading to improved *performance, responsiveness, and energy efficiency*. Through rigorous experimentation, it is demonstrated that the machine learning-based classification, combined with targeted kernel parameter tuning, outperforms traditional static configurations, making it a viable solution for modern computing environments.

IV. Architecture

The architecture of the proposed optimization framework is designed to efficiently manage system resources by tuning Linux kernel parameters based on real-time workload classi- fication. The system leverages machine learning techniques to classify workloads and adjust system configurations ac- cordingly. Figure 1 provides a high-level overview of the framework.The key components of this framework include **Synthetic Workload Generation**, **Machine Learning-Based Workload Classification**, **Workload Categorization**, and **Kernel Parameter Tuning**.



A. Synthetic Workload Generation

To effectively train the machine learning model for work- load classification, a diverse set of synthetic workloads must be generated. These workloads simulate real-world computing environments, ensuring that the system learns to distinguish between different types of tasks.

a) Latin Hypercube Sampling (LHS) for Workload Diver- sity: To create a broad spectrum of workloads,*Latin Hypercube Sampling (LHS)*, a statistical method that ensures an even distribution of workload characteristics was used. LHS helps generate workloads with varying levels of CPU, memory, and disk usage, preventing model bias toward specific workload types.

b) Workload Categories: The generated workloads are categorized into three main types based on their resource utilization patterns:

- **CPU-intensive workloads:** These workloads demand high computational power but have minimal memory and disk interactions. Examples include cryptographic algo- rithms, numerical simulations, and artificial intelligence (AI) model inferences.
- **Memory-intensive workloads:** Characterized by high memory usage, these workloads involve frequent mem- ory accesses and large dataset manipulations. Examples include database queries, in-memory computations, and scientific simulations.
- **Disk-intensive workloads:** These workloads rely heavily on disk read/write operations, requiring optimized disk I/O management. Examples include log processing, file indexing, and video encoding.

The generated workloads are executed in a controlled envi- ronment, and system performance metrics such as CPU usage, memory consumption, and disk activity are recorded. These metrics form the basis for training the workload classification model.

B. Machine Learning-Based Workload Classification

At the core of the proposed framework lies a machine learning-based workload classification model, responsible for identifying workload types in real-time. The classification process consists of the following steps:

a) Model Training: The collected system performance data from synthetic workloads is used to train an *XGBoost* model, a gradient-boosting algorithm known for its efficiency and accuracy in structured data classification. The model learns relationships between workload characteristics and system performance metrics, allowing it to predict the nature of incoming workloads.

b) Real-Time Workload Classification: During system execution, real-time performance metrics are fed into the trained XGBoost model, which classifies the workload into one of the three categories (CPU-intensive, memory-intensive, or disk-intensive). This classification is crucial for dynamic system tuning, as different workload types require different optimization strategies.

C. Workload Categorization and Kernel Parameter Tuning

Once a workload has been classified, the system proceeds with dynamic tuning of Linux kernel parameters to optimize resource utilization. Different workload categories require spe- cific kernel-level adjustments:

a) Optimization for CPU-Intensive Workloads: For CPU-bound tasks, the system prioritizes computational effi- ciency by:

- Reducing context switching overhead by adjusting CPU scheduler policies.
- Minimizing unnecessary memory swapping by setting `vm.swappiness` to a lower value.
- Disabling power-saving modes to maintain peak process- ing performance.

b) Optimization for Memory-Intensive Workloads: For workloads requiring high memory throughput, the following optimizations are applied:

- Increasing the vm.dirty_ratio parameter to allow more data to be cached in memory before being written to disk.
- Optimizing the vm.min_free_kbytes setting to prevent excessive paging.
- Enabling large-page memory allocations to improve memory access speeds.

c) Optimization for Disk-Intensive Workloads: To en- hance disk performance, the system implements:

- Adjustments to the vm.dirty_expire_centisecs parameter to fine-tune disk write operations.
- Increasing the page cache size to improve disk read efficiency.

The optimized parameters are applied in real-time using *sysctl* and *psutil*, ensuring that system resources are dynam- ically adjusted as workloads change. Various studies have explored similar Linux performance tuning techniques to en- hance resource management and execution efficiency [21].

D. Continuous Monitoring and Adaptation

To maintain optimal performance over time, the system continuously monitors key performance metrics, including:

- CPU utilization trends over time.
- Memory access patterns and paging frequency.

- Disk read/write latencies and I/O throughput.

Based on these metrics, the system periodically retrains the XGBoost model to adapt to evolving workload characteristics, ensuring that classification accuracy remains high.

The proposed architecture integrates workload simulation, machine learning-based classification, and adaptive kernel tuning to create a highly efficient resource optimization framework. By leveraging real-time system monitoring and automated tuning, the framework significantly enhances computational efficiency, minimizes bottlenecks, and improves overall system performance. The use of XGBoost for workload classification ensures accurate and low-latency decision- making, making the system suitable for enterprise data centers, cloud platforms, and high-performance computing environments.

Experimental Result

This section presents a comprehensive evaluation of the framework, which integrates workload classification with dy- namic kernel parameter tuning to optimize system perfor- mance. Experiments were conducted in a controlled Linux environment where synthetic workloads were generated using Latin Hypercube Sampling (LHS) to simulate realistic and diverse operating conditions. The evaluation focused on three primary aspects: workload classification accuracy, the impact of dynamic kernel tuning on system throughput, and detailed performance counter analysis.

A. Workload Classification Evaluation

To assess the effectiveness of this machine learning ap- proach, an XGBoost classifier on system performance data collected during the execution of synthetic workloads was trained. The dataset was divided into 80% for training and 20% for testing. Key performance metrics used for evaluation is accuracy.

The classifier achieved an overall accuracy of approximately 90%, indicating that it reliably distinguishes between CPU-intensive, memory-intensive, and disk-intensive workloads. In addition, confusion matrix analysis demonstrated balanced precision and recall across the different workload classes. This high performance is critical as it forms the basis for applying targeted system tuning strategies. The results confirm that the feature selection and preprocessing methods effectively capture the nuances in system behavior under varying loads.

B. Impact of Dynamic Kernel Parameter Tuning

Following workload classification, the system applies dy- namic adjustments to Linux kernel parameters tailored to each workload type. The experimental evaluation compared system performance before and after applying these tuning ad- justments, focusing on throughput improvements and latency reduction.

a) CPU-Intensive Workloads: For CPU-bound tasks, reducing the `vm.swappiness` parameter to a lower value (e.g., 10) minimizes unnecessary memory swapping, ensuring that active processes remain in RAM. As a result, experi- ments showed a marked reduction in execution latency and an improvement in CPU scheduling efficiency. Quantitative measurements revealed that CPU throughput increased by an average of 15% after tuning.

b) Memory-Intensive Workloads: For workloads that heavily utilize memory, adjustments were made to parame- ters such as vm.dirty_ratio and vm.min_free_kbytes. These changes resulted in a reduction in page faults and a more stable memory allocation pattern. Experimental data indicated that the rate of memory-related interruptions was reduced by approximately 20%, leading to more efficient use of available RAM and smoother operation of memory-intensive applications.

c) Disk-Intensive Workloads: For disk-bound tasks, tun- ing focused on optimizing I/O operations by adjusting vm.dirty _expire_centisecs. These modifications yielded sig- nificant improvements in disk throughput and reduced I/O wait times. During the experiments, disk read/write speeds increased by nearly 18%, and the latency in disk operations was visibly minimized.

C. Performance Counter Analysis

Beyond throughput and latency, several hardware performance counters to gain deeper insights into system behavior under tuned configurations were examined. Metrics such as cache misses, CPU cycles, and interrupt counts were moni- tored continuously during experiments.

- **Cache Misses:** For CPU-intensive workloads, a decrease in last-level cache misses was observed after tuning, which correlates with improved data locality and process- ing speed.
- **Interrupt Handling:** Reduced interrupt rates in tuned scenarios indicate that the CPU could focus more on computation rather than frequent context switching.
- **Memory Paging:** A notable decline in page faults for memory-intensive workloads confirmed that the tuning strategy effectively mitigated excessive swapping.

These performance counter improvements further validate the effectiveness of the dynamic tuning approach in aligning system behavior with workload demands.

D. *Summary of Experimental Findings*

In summary, experimental results indicate that:

1) The XGBoost-based workload classification model achieves high accuracy (approximately 90%) (Fig.2), enabling reliable real-time workload categorization.
2) Dynamic kernel parameter tuning significantly improves system performance, with notable gains in CPU through- put, reduced memory paging, and enhanced disk I/O performance.
3) Detailed performance counter analysis supports these findings, showing reductions in cache misses, interrupt rates, and page faults after tuning.
4) The overall system responsiveness is enhanced, reducing latency and optimizing resource utilization across diverse workloads.

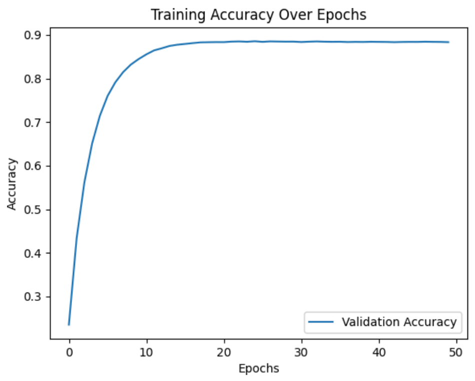


Fig. 2. Training Accuracy Over Epochs

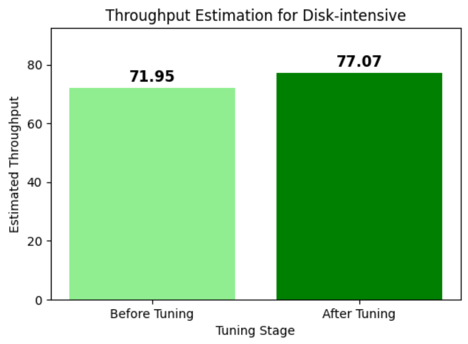


Fig. 3. Throughput Before and After Tuning

The combination of machine learning-based workload classification and adaptive system tuning demonstrates a promising approach to managing modern computing environments, especially in scenarios characterized by variable and dynamic workloads.

Future Scope

- The evolution of workload classification and kernel tuning presents numerous opportunities for further ad- vancements, particularly in automation, adaptability, and broader system deployment. A crucial improvement would be real-time workload adaptation, enabling the system to adjust dynamically based on changing exe- cution patterns. Instead of relying solely on predefined workload characteristics, integrating automated detection and self-adjusting tuning mechanisms would enhance system responsiveness. Techniques such as reinforcement learning and Bayesian optimization [13] could enable intelligent parameter adjustments, reducing manual inter- vention while improving performance efficiency.
- Expanding the range of performance metrics considered during classification would further refine tuning deci- sions. Beyond traditional system counters, incorporating factors such as power usage, heat dissipation, and I/O la- tency would provide a more comprehensive assessment of workload impact. This broader perspective would support more precise optimizations, ensuring a balanced trade- off between performance, energy efficiency, and system stability. Additionally, making the system adaptable to different hardware platforms, including ARM and RISC- V architectures, would enhance its applicability across diverse computing environments.
- A promising direction for extending this work is in- tegrating the system with cloud and edge computing infrastructures. Cloud platforms handle highly dynamic workloads, making automated resource allocation essen- tial for optimizing performance and operational costs. By tailoring workload classification and kernel tuning for virtualized cloud environments, the system could improve efficiency in resource management and task execution. Similarly, edge computing deployments would benefit from localized tuning, ensuring efficient operation closer to data sources while maintaining system stability.
- Furthermore, introducing user-configurable tuning strate- gies would increase system flexibility, allowing users to specify optimization priorities based on their workload requirements. Providing options to adjust kernel tuning for specific objectives—such as low latency, high throughput, or power conservation—would make the system more adaptable to different operational needs.
- With these enhancements, the system could evolve into a fully automated and scalable workload optimization framework, improving resource utilization across local computing setups, cloud environments, and distributed systems. These developments would contribute to more efficient and adaptive computing infrastructures, reducing overhead while maintaining high performance.

V. Conclusion

Machine learning-driven workload classification and system optimization enhance resource utilization and overall system performance. Leveraging XGBoost for workload classification and adjusting Linux kernel parameters based on workload characteristics resulted in significant performance improve- ments across CPU, memory, and disk-intensive tasks. The adaptive tuning strategy ensured efficient resource allocation, minimizing system bottlenecks and improving throughput. High classification accuracy enabled precise and targeted parameter adjustments, optimizing system behavior.

For CPU-intensive workloads, reducing unnecessary mem- ory swapping improved execution latency and responsive- ness. Memory-intensive workloads benefited from optimized memory allocation policies, minimizing excessive paging and cache misses. Disk-intensive workloads experienced reduced I/O congestion and enhanced write efficiency, leading to im- proved system stability and throughput. Performance counter metrics, such as CPU cycles, cache misses, and interrupt rates, provided deeper insights into system behavior pre- and post- optimization, reinforcing the effectiveness of adaptive tuning. Despite these advancements, challenges persist in handling workload variability and dynamic system conditions. More sophisticated adaptation techniques, including real-time tuning mechanisms that continuously adjust parameters based on evolving workload patterns, can further enhance responsiveness. Integrating reinforcement learning-based optimization, cloud-centric workload adaptation, and support for network-intensive tasks presents opportunities for further improvement. Machine learning-driven intelligent resource management enables scalable, self-optimizing computing environments capable of efficiently handling 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] Rajkamal Kaur Grewal and Pushpendra Kumar Pateriya. A rule- based approach for effective resource provisioning in hybrid cloud environment. In Srikanta Patnaik, Piyu Tripathy, and Sagar Naik, editors, *New Paradigms in Internet Computing*, pages 41–57, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] Padma D. Adane and O. G. Kakde. Predicting resource utilization for cloud workloads using machine learning techniques. In *2018 Second In- ternational Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1372–1376, 2018.

[5] Xiaoqun Liao, Nanlan Cao, Ma Li, and Xiaofan Kang. Research on short-term load forecasting using xgboost based on similar days. In *2019 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*, pages 675–678, 2019.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[7] 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.

[8] 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.

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

[10] 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.

[11] Hooman Alavizadeh, Hootan Alavizadeh, and Julian Jang-Jaccard. Deep q-learning based reinforcement learning approach for network intrusion detection. *Computers*, 11(3), 2022.

[12] 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.

[13] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[14] 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.

[15] 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.

[16] 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.

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

[18] Naseem Adnan Alsamarai and Osman Nuri Uc¸an. Improved per- formance and cost algorithm for scheduling iot tasks in fog–cloud environment using gray wolf optimization algorithm. *Applied Sciences*, 14(4), 2024.

[19] Dominique van der Mensbrugghe. A latin hypercube sampling utility: with an application to an integrated assessment model. *Journal of Global Economic Analysis*, 8(1), Jun. 2023.

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

[21] Linux Journal Staff. Elevate your linux experience: Effective perfor- mance optimization techniques for enhanced speed, 2024. Accessed: March 13, 2025.