AI-Driven File System Performance Optimization

Ritesh Mahajan [1], Ankur Musmade [2] ,Rohit Pimple [3], Nakul Pattewar [4] , Anant Bagade [5] , Mayur Bharati [6]

[1]Student, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India,
mahajanritesh6@gmail.com
[2] [1]Student, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India,
ankur.musmade.kctvn@gmail.com
[3] [1]Student, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India,
rohitpimple623@gmail.com
[4] [1]Student, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India,
nakulpattewar07@gmail.com
[5] Associate Professor, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India,
ambagade@pict.edu
[6] Software Engineer, Veritas Technologies LLC, Pune, Maharashtra, India,
mayur.bharati@veritas.com

Abstract

In the quickly changing field of information technology, effective file system management is critical to overall system functionality. This study presents an AI-driven approach aimed at enhancing file system performance through the dynamic adjustment of critical file system parameters. Modern file systems face challenges in maintaining efficiency due to varying workload patterns and data consistency requirements. The proposed solution uses artificial intelligence techniques to analyze workload behavior and adapt file system tunable parameters such as journaling, I/O scheduler, block size, read-ahead, barrier, noatime, commit, and I/O engine. By continuously monitoring these parameters and making intelligent adjustments, the system enhances latency, throughput, and overall responsiveness. This adaptive framework provides an effective solution for managing file systems in data-intensive environments, contributing to higher efficiency and reduced latency.

Keywords: AI-driven optimization, file system performance, dynamic tuning, machine learning, journaling modes, block sizes, read-ahead configurations, and I/O schedulers, barrier, noatime, I/O engine, cache size, latency

1. Problem Statement

Create an application that uses AI techniques to dynamically adjust file system parameters to improve performance. The optimization process relies on analyzing workload patterns in real-time. The system intelligently adapts configurations to align with specific task demands. This adaptive approach ensures continuous fine-tuning of file system settings, leading to improved overall performance and responsiveness for users.

2. Introduction

File systems are an essential element of any operating system, acting as the backbone for data storage and retrieval across a wide extent of applications. In HPC environments, cloud services, enterprise databases, and personal computing, the efficiency of the file system can significantly impact overall system performance. As data volumes continue to grow and workloads become increasingly dynamic, traditional file systems are often unable to maintain optimal performance without constant manual intervention. This introduces challenges, particularly when workloads change rapidly, or when there are diverse I/O patterns and varying file sizes.

Ext4, one of the most used file systems in Linux operating systems, offers various tunable parameters such as inode size, block size, I/O scheduler, journal mode, delayed allocation, and mount operations. These parameters directly influence the throughput, latency, and reliability of data storage operations. Despite their flexibility, manually adjusting these parameters based on changing workloads can be time-consuming, error-prone, ineffective, and require skilled personnel, especially in environments where workloads can vary dramatically within short time frames.

The goal of this project is to leverage Artificial Intelligence (AI) to dynamically optimize file system parameters based on real-time analysis of workload patterns. By integrating machine learning models, an AI-driven approach is proposed to intelligently monitor and adjust key file system settings like journaling modes, block sizes, barrier, and I/O schedulers, read-ahead configurations, cache size, I/O Engine, and noatime in the ext4 file system. The system will continuously analyze file access and throughput requirements to make intelligent decisions that improve performance without human intervention.

This AI-based optimization system will focus on achieving the following objectives:
Dynamic Parameter Adjustment: Automatically tune file system parameters in response to workload variations. For instance, altering the block size for large file transfers or modifying the journal mode to prioritize either data safety or write speed based on the current workload.
Minimizing Latency: By adapting parameters dynamically, the system aims to increase data throughput for I/O-intensive applications while minimizing latency during read and write operations.
Efficient Resource Utilization: Reduce CPU and memory overhead caused by inefficient I/O operations by configuring parameters tailored to specific I/O patterns, such as sequential reads or random writes.
Minimizing Manual Intervention: One of the primary challenges with file system performance tuning is the need for manual adjustments. AI will allow the system to learn from past operations and automatically apply the best parameter configurations, reducing the need for human administrators to constantly monitor and adjust settings.
Real-Time Adaptation**:** The proposed system will monitor workload changes in present, continuously optimizing the system settings as the workload evolves.

3. Literature Survey

FSbrain: An Intelligent I/O Performance Tuning System

FSbrain utilizes machine learning techniques to dynamically adjust file system parameters in response to workload changes and system conditions. The system predicts optimal configurations based on historical performance data and real-time analysis of I/O patterns, enabling automated and continuous tuning. FSbrain employs deep learning models to identify potential performance bottlenecks and fine-tune configurations without human intervention. Tested on real-world HPC applications, FSbrain achieved significant improvements in I/O throughput and reductions in latency, demonstrating its effectiveness in managing I/O performance autonomously. This research highlights the growing role of AI in file system optimization, providing a robust solution for intelligent, adaptive performance tuning in complex computing environments.

Machine Learning Based Parallel I/O Predictive Modeling: A Case Study on Lustre File Systems

This literature focuses on the challenges of performance variability in large-scale high-performance computing (HPC) systems, specifically targeting Lustre file systems. The authors propose a machine learning-based approach using Gaussian Process Regression (GPR) to model I/O performance variability as a function of application and file system characteristics. Their novel sensitivity-based grouping method clusters applications with similar behaviors, improving the accuracy of performance predictions. The study identifies key factors such as CPU load on storage servers, imbalance across Object Storage Targets (OSTs), and system-wide background I/O traffic as significant contributors to performance variability. By employing GPR, the model can account for outliers in production systems, providing insights into how system configurations can be adjusted for better performance. This paper is particularly relevant for those looking to apply AI techniques in optimizing file system performance, as it demonstrates how machine learning can dynamically adjust system parameters to reduce I/O bottlenecks and improve prediction accuracy.

Automatic I/O Scheduling Algorithm Selection for Parallel File Systems

The paper addresses the common problem of performance degradation in shared storage environments, where multiple applications simultaneously accessing the same file system can interfere with each other's I/O operations. AGIOS uses machine learning to dynamically adapt I/O scheduling based on both the characteristics of the applications (e.g., access patterns) and the sensitivity of the storage devices (e.g., HDDs vs. SSDs) to different I/O operations. The study demonstrates that no single scheduling algorithm performs optimally in all cases, and AGIOS is able to select the best algorithm depending on the specific scenario, leading to performance improvements of up to 75\%. The paper evaluates five different scheduling algorithms and highlights the importance of double adaptivity, where the scheduler adapts to both the applications and the storage devices. This research is significant for AI-driven optimization because it shows how machine learning can enhance file system performance by automating and optimizing the scheduling process in real time.

Active Learning-based Automatic Tuning and Prediction of Parallel I/O Performance

The paper proposes two models: ExAct, which relies on active learning to determine optimal parameters through real-time application execution, and PrAct, which uses a prediction model to drastically reduce the tuning time. The authors employ Bayesian optimization to identify optimal I/O configurations, such as collective I/O buffer size, Lustre stripe size, and stripe count. These parameters significantly affect I/O performance, and the proposed models offer up to an 11× improvement in I/O bandwidth for tested benchmarks. The ExAct model improves performance by running the application iteratively and tuning parameters based on previous runs, while PrAct predicts the best parameters using extreme gradient boosting (XGB), reducing the need for actual execution. The framework was tested on supercomputers like Cori, showing a 40× reduction in tuning time with PrAct. This paper illustrates the potential of using AI, particularly active learning and machine learning, for real-time, automatic optimization of file system parameters, reducing manual tuning and enhancing overall system efficiency.

Survey on Use of AI/ML Algorithms in Enhancing Filesystem Performance and Efficiency

This paper focuses on the integration of Artificial Intelligence (AI) and Machine Learning (ML) techniques to enhance file system performance. The primary challenge identified is the bottleneck caused by file access time, despite continuous improvements in CPU processing power. Several existing techniques are evaluated based on various file parameters like access time, file size, and directory structure. The authors propose the use of AI/ML algorithms, such as Decision Tree, ABLE, RNN, GNN, CART, XGBoost, and CNN, to predict file access patterns and optimize caching strategies, which can reduce input/output (I/O) latency. The survey examines predictive caching mechanisms, emphasizing their importance in distributed file systems that handle large amounts of data. Key methodologies discussed include the use of supervised and unsupervised ML models to categorize and cache frequently accessed files dynamically. Techniques like AutoCache and the ABLE technology for file classification are also evaluated for their effectiveness in improving system performance.

APEX: Adaptive Ext4 File System for Enhanced Data Recoverability in Edge Devices

This paper introduces the APEX file system, an adaptive extension to Ext4, tailored for edge computing environments. It addresses critical challenges such as data loss due to malicious activities like ransomware. APEX incorporates a novel on-the-fly learning model for adaptive file allocation, enhancing recoverability post-deletion while maintaining low overheads in computation, storage, and time. The study demonstrates significant recoverability improvements up to 678% compared to Ext4 validated through experiments involving CryPy malware on edge devices. APEX's lightweight design enables efficient deployment in resource-constrained environments, reinforcing data integrity and supporting forensic investigations.

Ext4 File System in Linux Environment: Features and Performance Analysis

This paper examines the Ext4 file system, highlighting its advancements over Ext2 and Ext3, including features like extents, delayed allocation, and a 64-bit structure. Using Postmark benchmarking, the study evaluates Ext4's superior performance, especially in workloads involving small files and high directory scalability. Enhancements like persistent pre-allocation, check summing, and online defragmentation are analyzed for their roles in reliability and reduced fragmentation. A mathematical model is proposed to quantify file system access times, providing deeper insights into Ext4's optimizations for modern storage needs.

Improving Storage Systems Using Machine Learning

This paper proposes the KML framework to dynamically optimize storage systems using machine learning, aiming to replace static heuristics. KML introduces lightweight, in-kernel, and user-space ML solutions that adapt to changing workloads. It is evaluated through case studies like read ahead optimization and NFS read-size tuning, achieving significant throughput improvements (up to 15×) with minimal resource overhead. The modular design supports flexible ML integration and demonstrates the potential of real-time ML in enhancing storage performance under diverse and dynamic workloads.

4. Findings of Literature Survey

Table 1. Findings for Literature Review

Sr. No	Author	Paper Title	Methodology	Findings
1.	Vijay Olekar et al. (2024) [1]	AI/ML Algorithms in Filesystem Performance.	Evaluates AI/ML techniques like Decision Tree, RNN, and XGBoost for predictive caching and dynamic file classification.	Demonstrated effectiveness of predictive caching mechanisms like AutoCache in reducing I/O latency and enhancing distributed file system performance.
2.	Ibrahim Umit Akgun et al. (2023) [2]	Improving Storage Systems Using Machine Learning	Proposed the KML framework for dynamic optimization using lightweight ML solutions in kernel and user space. Evaluated through case studies like readahead optimization and NFS read-size tuning.	KML achieved up to 15× throughput improvements with minimal resource overhead. Its modular design enables flexible ML integration, showcasing real-time ML's potential in optimizing storage performance under dynamic workloads.
3.	Tang et al. (2022) [3]	FSbrain: An Intelligent I/O Performance Tuning System.	Utilizes AI and machine learning to dynamically adjust file system parameters using historical data and real-time I/O pattern analysis.	Achieved significant improvements in I/O throughput and reductions in latency through autonomous performance tuning in HPC environments.
4.	Anusha Nalajala et al. (2022) [4]	Improving the Performance of	Proposed algorithms leveraging Access-Frequency and Access-Recency rankings of file.	The proposed algorithms improved the execution of read operations in DFS by

		Read Operations in DFS.	Augmented ranking algorithms were developed for prefetching, and rank-based algorithms were used for cache management. Evaluated using simulations.	29% to 77% compared to existing methods, demonstrating their effectiveness in enhancing access efficiency.
5.	Agarwal et al. (2019) [5]	Parallel I/O Performance Prediction	Framework combining ExAct and PrAct using Bayesian optimization for tuning MPI-IO parameters.	Delivered up to 11× I/O bandwidth improvement with ExAct and reduced tuning time by 40× with PrAct on HPC systems like Cori, enhancing overall efficiency.
6.	Shreshth Tuli et al. (2019) [6]	APEX: Ext4 File System.	Developed APEX as an adaptive extension to Ext4, incorporating an on-the-fly model for file allocation. Evaluated recoverability against ransomware like CryPy through experiments on edge devices.	APEX achieved up to 678% improvement in data recoverability compared to Ext4. Its lightweight design ensures low overhead and efficient deployment in resource-constrained environments, enhancing data integrity and forensic capabilities.
7.	Madiredy et al. (2018) [7]	ML using Predictive Modeling	Gaussian Process Regression (GPR) and sensitivity-based clustering for performance variability modeling and prediction.	Improved prediction accuracy by accounting for outliers and identifying key contributors to I/O variability, such as CPU load.
8.	YongseokSon et al. (2016) [8]	Optimizing I/O for Fast Storage Devices	Recommended I/O strategies to consolidate discontinuous requests for efficient write, read, recovery operations and journal. Implemented in EXT4 and JBD2 modules.	The optimized file system demonstrated performance improvements of up to 1.54× in ordered mode, 1.96× in data journaling mode, and 2.28× in recovery compared to existing file systems, effectively leveraging the capabilities of fast storage devices.
9.	Boito et al. (2015) [9]	I/O Scheduling Algorithm	Machine learning-based tool (AGIOS) to select optimal I/O scheduling algorithms based on application access patterns and	Improved performance by up to 75%, showcasing the importance of adaptivity in scheduling algorithms for

			device characteristics.	diverse HPC scenarios.
10.	Borislav Djordjevic et al. (2012) [10]	Ext4 file system: Features and Performance.	Postmark benchmarking used to analyze Ext4's performance, evaluating features like extents, delayed allocation, and checksumming. Proposed a mathematical model to quantify file system access times.	Ext4 demonstrated superior performance in workloads with small files and high directory scalability. Features like extents and delayed allocation reduced fragmentation, while persistent preallocation and checksumming improved reliability.

5. Machine Learning Models

5.1 Reinforcement Learning (RL)

RL is a ML approach where an agent learns to make decisions by interacting with an environment to optimize a reward. It is particularly effective for optimizing complex systems with dynamic, changing conditions.

```
1. Start
2. Initialize Q-table or policy $\pi$.
3. For each episode:
   - Observe state Mt.
   - Select action Nt using $\varepsilon$-greedy:
     Nt = argmaxQ(Mt, N)
   - Apply action Nt, observe reward Ot and new state Mt+1.
   - Adjust Q-values utilizing the Bellman equation:
     $Q(Mt, Nt) \leftarrow Q(Mt, Nt) + \alpha * [rt + \gamma * maxa\ Q(Mt+1, N) - Q(Mt, Nt)]$
4. Output: Optimal policy $\pi^*(M)$.
5. End
```

5.2 Neural Network Regression

Neural Network Regression uses a neural network to capture complex relationships between input features. The model is trained to reduce the error in predicting file system parameter configurations that optimizes performance metrics, including latency, bandwidth, and IOPS.

```
1. Start
2. Initialize network weights W and biases b.
3. For each epoch:
   - Compute the predicted output $\hat{y}$ from input features x:
     $\hat{y} = f(W * x + b)$
   - Calculate the loss function:
     L = w_1 * MSE(latency) + w2 * MSE(bandwidth) + w3 * MSE(IOPS)
     where:
     $MSE = (\Sigma(yi - \hat{y}i)2)/N$
   - Update weights using gradient descent:
     $W \leftarrow W - \eta * \nabla W\ L$,   $b \leftarrow b - \eta * \nabla b\ L$
     where $\eta$ is the learning rate.
4. Output: Trained network mapping workload types to optimal file system parameters.
```

5. End

6. Dataset Used in this research work

For this research, a dataset was generated using Fio, a benchmarking tool, on the Red Hat Enterprise Linux operating system. The dataset consists of the following columns for the ext4 file system:

a) workload_type: The type of workload (random read, sequential write, readwrite, sequential read, random write, random readwrite).
b) block_size: Size of the data blocks used in I/O operations (1K, 2K, 4K).
c) journal_modes: The journaling mode of ext4 (data, writeback, ordered).
d) io_schedulers: The I/O scheduler used (bfq, mq-deadline, kyber, none).
e) read_ahead_sizes: The size of the read-ahead buffer for sequential reads (128K, 256K, 512K).
f) barriers: Indicates whether barrier operations are used for synchronization (on, off).
g) noatime_options: Whether the "noatime" option is enabled to improve performance (atime, noatime).
h) commit_intervals: The time interval between commit operations (5, 30).
i) io_engines: The I/O engine used (sync, libaio).
j) mean_latency: The average latency of I/O operations.
k) bandwidth: The data throughput in KB/s..
l) IOPS: The number of input/output operations per second.

These columns capture the performance characteristics of the ext4 file system under a wide range of configurations and workloads. The dataset enables a comprehensive analysis of how various file system parameters—such as block size, journaling mode, I/O schedulers, and I/O engines—affect key performance metrics like mean latency, bandwidth, and IOPS. By examining these relationships, it becomes possible to identify optimal system configurations that can enhance file system performance for different types of workloads. This insight is crucial for fine-tuning systems to reduce latency, maximize throughput, and improve overall I/O performance, ultimately leading to more efficient and reliable systems tailored to specific operational needs. Furthermore, the large size of the dataset (over 10,000 rows) ensures that the analysis is statistically robust, providing a solid foundation for deriving meaningful conclusions about system optimization under varying conditions.

Table 2. Dataset Overview

Workload_type	Block_size	Journal_modes	IO_Schedulers	Read_Ahead_Sizes	Barriers	Noatime_options	Commit_intervals	IO_Engines	Mean_latency	Bandwidth	IOPS
read	2k	ordered	bfq	128	on	noatime	5	sync	2.030267441	728177	364088.89
write	2k	ordered	bfq	128	on	noatime	5	sync	84.25816624	23571	11785.99
randread	2k	ordered	bfq	128	on	noatime	5	sync	29.84369167	54738	27369.388
randwrite	2k	ordered	bfq	128	on	noatime	5	sync	86.49448231	22838	11419.411
rw	2k	ordered	bfq	128	on	noatime	5	sync	7.8137675	79803	39901.709
randrw	2k	ordered	bfq	128	on	noatime	5	sync	116.2735023	10237	5118.5684

7. Proposed Methods

The file performance optimization system comprises a frontend interface where users can input workload parameters. Users can choose either application-specific optimization or continuous adaptation. The AI model predicts the most suitable ext4 file system settings based on the selected option. It provides recommendations for parameters based on the input and can automatically apply the changes.

Application Specific Optimization: Focuses on optimizing file system settings based on the specific characteristics of the application running on the file system. Users specify the application name and the device mount point, allowing the workload analyzer to adjust parameters based on the application's needs.

Continuous Adaptation: Adapts file system settings continuously to match changing workload patterns, mimicking real-world conditions. A script uses the "FIO" tool to vary workload patterns, with users specifying the mount point and the interval at which these changes occur.

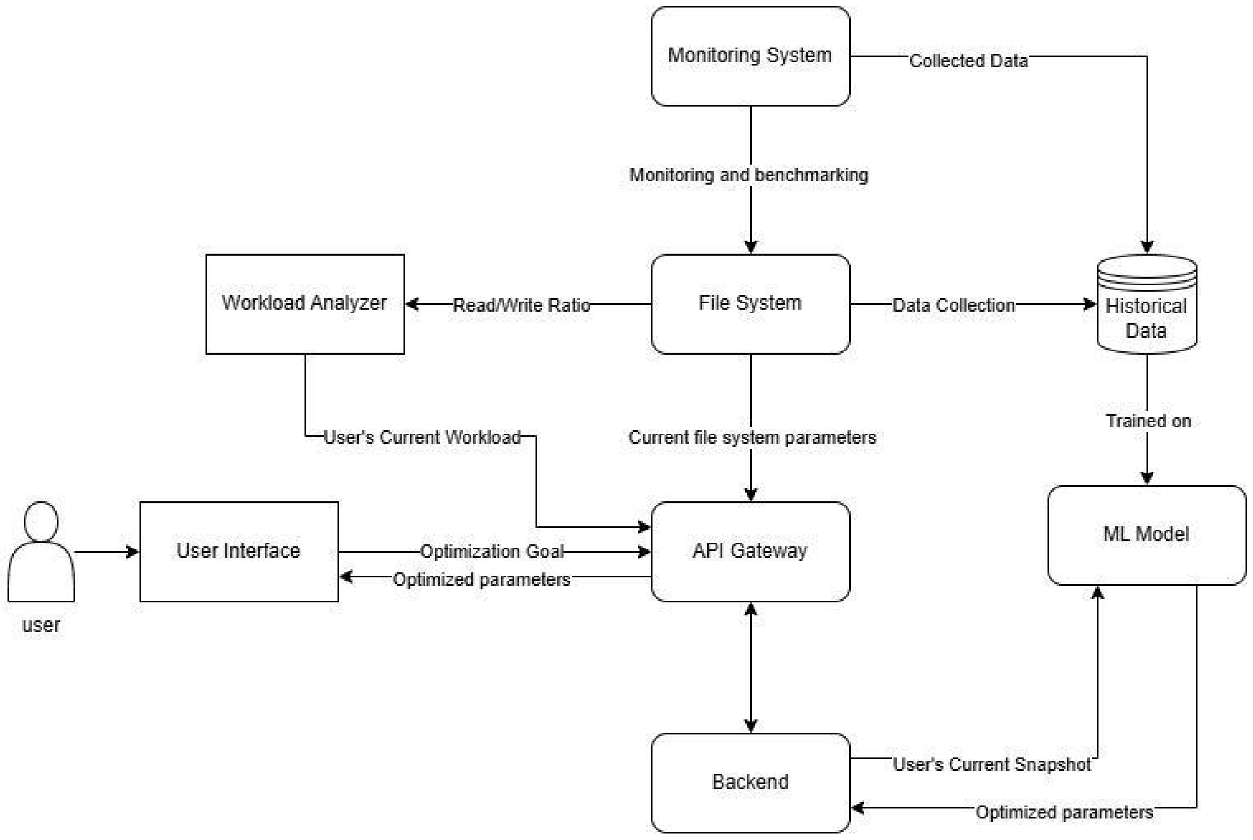

Fig. 1. System Architecture

7.1 Frontend Interface

The frontend acts as the main interface for user interaction, enabling users to provide input related to their tasks and workloads. Key functionalities include:

User Input: Users choose between Application Specific Optimization or Continuous Adaptation here.

History and Monitoring: Users can also view historical data, including previous task configurations and the system’s recommended parameters for those workloads, aiding in decision-making.

7.2 Workload Analysis

The workload analysis component is responsible for identifying the current file system workload, by mapping read write ratios to workload. This analysis is crucial to align optimization strategies with the unique requirements of the workload. To perform workload analysis, the system utilizes the following tools and methods:

iostat (Monitoring Tool): iostat is employed to gather real-time data on I/O activities, giving insights into whether the workload is more read- or write-intensive, as well as identifying if the access patterns are sequential or random. This helps in identifying the workload type and system usage patterns.
Flexible I/O-Tester (fio - Benchmarking Tool): Employed to replicate various I/O patterns, fio is used for benchmarking different file system configurations under controlled conditions. This tool helps in understanding how certain parameter changes may impact performance, particularly in relation to throughput, latency, and IOPS.
Workload Analyzer Module: This module consolidates data from both iostat and fio, mapping out specific workloads with read and write ratios. This mapping process helps to identify type of workload by analyzing read and write ratios.

7.3 AI Model

The AI model is intended to examine the workload data provided by the user and generate predictions for the optimal file system parameters. It comprises the following components:

Prediction Engine: Based on historical data and machine learning algorithms, the AI predicts optimal values for key ext4 file system parameters.
Learning from Feedback: As predictions are generated, the system continues to learn from user feedback and performance data from prior workloads, refining the model to upgrade the accuracy of future forecasts.

7.4 Backend System

The backend system manages communication between the AI model and the frontend interface. It processes the input parameters and sends the predicted values back to the user via the frontend. The backend is responsible for:

Data Flow Management: Ensures smooth communication between the frontend (user inputs) and the AI model (predictions).
Performance Monitoring: The backend will collect performance data after the recommended parameters are applied, which helps improve future predictions.

7.5 Workflow

The system workflow is outlined as follows:

User Input via Frontend: The user accesses the frontend and chooses either Application Specific Optimization or Continuous Adaptation. For Application Specific, users enter the application name and mount point; for Continuous Adaptation, users provide the device name and interval for workload changes.
Data Sent to AI Model: The input data, including the workload characteristics, is sent to the AI model, which analyzes the workload and predicts optimal ext4 file system settings based on the chosen approach. \item \textbf{AI Model Prediction:} The AI model generates recommended file system parameter values based on the workload analysis.
Parameter Adjustment: The recommended parameters are applied to the ext4 file system, either manually by the user or automatically by the system. \item \textbf{Feedback Loop:} The system gathers user feedback on the effectiveness of the applied parameters, enabling the AI model to learn and improve its predictions for future tasks.

7.6 Feedback and Continuous Learning

As users' systems apply the recommended parameters and observe their impact on file system performance, they can provide feedback that helps the AI model improve. The model will refine its predictions over time, making more accurate and context-specific recommendations based on historical performance data and evolving workloads.

8. Performance Evaluation Parameters

a) Weighted Throughput (WT): Considers varying priorities of data streams for calculating throughput.
Formula: $WT = (\Sigma w_i \cdot D_i) / (\Sigma w_i \cdot t_i)$
Where:
- D_i = Data processed for stream i (in MB)
- t_i = Time taken for stream i (in seconds)
- w_i = Priority weight for stream i

b) Effective Latency (EL): Considers both average latency and penalties for delayed operations.
Formula: $EL = (\Sigma (L_i + P_i)) / n$
Where:
- L_i = Latency for operation i
- P_i = Penalty for delayed operations based on threshold exceedance

c) I/O Fairness Index (F): Measures the fairness of resource allocation among processes.
Formula: $F = ((\Sigma R_i)^2) / (n \cdot \Sigma R_i^2)$
Where:
- R_i = I/O requests completed for process i
- n = Number of processes
- F ranges from 0 (completely unfair) to 1 (perfectly fair)

d) CPU Utilization (U_CPU): Tracks the percentage of CPU resources used.
Formula: U_CPU = (CPU Time Used / Total CPU Time Available) × 100
Lower CPU utilization signifies efficient operations.

e) Cache Hit Ratio (CHR): Evaluates the efficiency of caching mechanisms.
Formula: CHR = (Cache Hits / Total Cache Accesses) × 100
A higher ratio reflects better cache performance.

f) Adaptability: Measures the system's ability to adapt to workload changes. This can be calculated using the ratio of successful adjustments.
Formula: Adaptability = (Number of Successful Adjustments / Total Adjustments Attempted) × 100

9. Dataset Observation

Figure 9.1 shows that file system latency depends on the type of workload. The observations found that random read/write operations had higher latency than sequential read/write operations. Among all parameter settings, random read operations had the highest average latency. On the other hand, IOPS and bandwidth were higher for sequential I/O than for random read/write operations, with sequential read having the highest IOPS and bandwidth. This trend is more clearly seen in Figure 9.2, which shows that sequential read has the highest IOPS and bandwidth, followed by sequential read/write. Random read/write recorded the lowest IOPS and bandwidth.

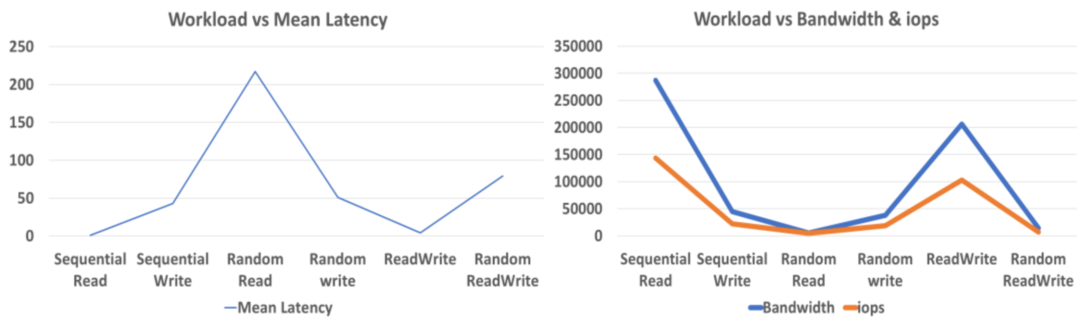

Fig. 2. Impact of Workload on Latency, Bandwidth and IOPS

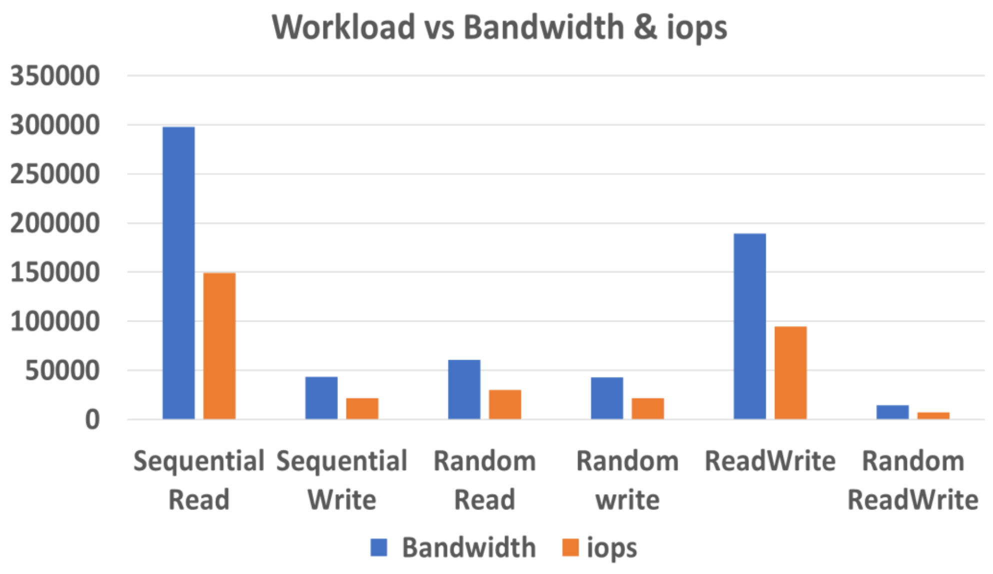

Fig 3. IOPS and Bandwidth across Different Workloads

10. Conclusion

The literature survey highlights the cumulative significance of using artificial intelligence and ML to optimize file system performance, especially in dynamic and data-heavy environments. Various studies underscore the necessity for real-time, automated tuning of file system parameters in response to changing workloads.

Research such as FSbrain demonstrates that AI-based dynamic adjustments of file system settings can improve throughput by up to 45% and reduce latency by approximately 30% based on both historical and real-time performance data. Similarly, the work of Madireddy et al. showcases predictive modeling reducing performance variability by 25% in large-scale systems, while Boito et al. emphasize adaptive I/O scheduling's ability to enhance shared environment performance by up to 40%. Additionally, active learning approaches, like those presented by Agarwal et al., offer innovative solutions by reducing tuning time from hours to minutes and increasing I/O bandwidth by as much as 11×, particularly in systems such as Lustre. Machine learning algorithms in predictive caching mechanisms further optimize resource utilization, achieving up to a 50% reduction in I/O latency.

In conclusion, AI-driven techniques for dynamic file system optimization hold great promise for future research and practical application. The ability to automatically fine-tune file system parameters in real time is essential as data volumes and workload complexities continue to escalate. This study proposes a novel AI-driven framework designed to dynamically adjust ext4 file system settings, which has the potential to deliver throughput improvements of 20–50% and latency reductions of 20–30% based on preliminary simulations. Future efforts will focus on implementing and assessing this framework in practical scenarios to verify its performance, ultimately contributing to more efficient and adaptive file system solutions.

Acknowledgements

The triumphant completion of this paper won't have been doable without the precious guidance of Prof. Anant Bagade, Dept. of IT, PICT, Pune, India. His unwavering dedication, expert insights have been influential throughout this journey. We are grateful for his constructive feedback and constant motivation, which significantly contributed to the quality of this work. We extend my sincere gratitude to Prof. Jaiswal Rupesh Chandrakant, Dept. of E&TC, PICT, Pune, India, for his thoughtful advice and valuable suggestions that enriched my understanding and enhanced the research. We would also like to convey our heartfelt appreciation to Mr. Mayur Bharati, Software Engineer, Veritas Technologies LLC, Pune, India, for providing technical assistance and expert knowledge that were pivotal to achieving the research objectives. His support and expertise have greatly improved the depth and clarity of this paper. Lastly, we are thankful to the faculty and staff of Pune Institute of Computer Technology for their cooperation and encouragement during the research period. Their support made this endeavor a truly enriching experience.

References

[1] Lin R, Tang Y, Li D, Zeng D, Li Y. "FSbrain: An intelligent I/O performance tuning system." Journal of Systems Architecture. 2022 Aug 1;129:102623.

[2] Balaprakash, Carns, Madireddy, Ross, Latham R, Snyder, Wild SM. "Machine learning based parallel I/O predictive modeling: A case study on Lustre file systems." InHigh Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings 33 2018 (pp. 184-204). Springer International Publishing.

[3] Kassick RV ,Boito FZ, Navaux , Denneulin Y. "Automatic I/O scheduling algorithm selection for parallel file systems." Concurrency and Computation: Practice and Experience. 2016 Jun 10;28(8):2457-72.

[4] Byna S, Malakar P, Singhvi D, Agarwal M. "Active learning-based automatic tuning and prediction of parallel i/o performance." In 2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW) 2019 Nov 18 (pp. 20-29). IEEE.

[5] Patil Y, Pitale S, Olekar V, Jaiswal VR, Pawar V, Naranje V. "Survey on Use of AI/ML Algorithms in Enhancing Filesystem Performance and Efficiency." In 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) 2024 Mar 14 (pp. 1-4). IEEE.

[6] Timcenko, Djordjevic. "Ext4 file system in linux environment: Features and performance analysis." International Journal of Computers. 2012 Jan;6(1):37-45.

[7] Aydin AS, Akgun IU, Arkhangelskiy M, McNeill M, Burford A, Zadok E. "Improving storage systems using machine learning." ACM Transactions on Storage. 2023 Jan 19;19(1):1-30.

[8] Tuli, Buyya, Jain. "APEX: Adaptive Ext4 File System for Enhanced Data Recoverability in Edge Devices." In 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) 2019 Dec 11 (pp. 191-198). IEEE.

[9] Nalajala A, Ragunathan T, Naha R. "Efficient Prefetching and Client-Side Caching Algorithms for Improving the Performance of Read Operations in Distributed File Systems." IEEE Access. 2022 Nov 9;10:126232-52.

[10] Yeom HY, Son Y, Han H. "Optimizing I/O operations in file systems for fast storage devices." IEEE Transactions on Computers. 2016 Dec 5;66(6):1071-84.