IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Adaptive RAM Optimization Using Machine Learning

Enhancing System Performance and Application Responsiveness

[1]Prof. Neha Bhagwat, [2]Saakshi Kobarne, [3]Vaishnavi Sapkal, [4]Aishwarya Karande
[1]Professor, [2]Student, [3]Student, [4]Student
[1]Department of Computer Engineering,
[1]Nutan Maharashtra Institute of Engineering and Technology, Pune, India

Abstract : The increasing complexity of modern computing systems demands efficient memory management to enhance performance and responsiveness. The existing SysMain (Superfetch) service in Windows preloads applications to improve launch times but suffers from high RAM and disk usage, making it inefficient for low-memory systems. This paper proposes a machine learning-based RAM optimization system that dynamically predicts and preloads only necessary applications. By leveraging behavioral data, adaptive algorithms, and real-time monitoring, the proposed system optimizes memory utilization, reducing unnecessary resource consumption while enhancing user experience. The study evaluates the system's effectiveness across different hardware configurations (SSD vs. HDD, low-RAM vs. high-RAM) and operating environments, demonstrating its superiority over traditional preloading mechanisms in terms of performance, responsiveness, and efficiency.

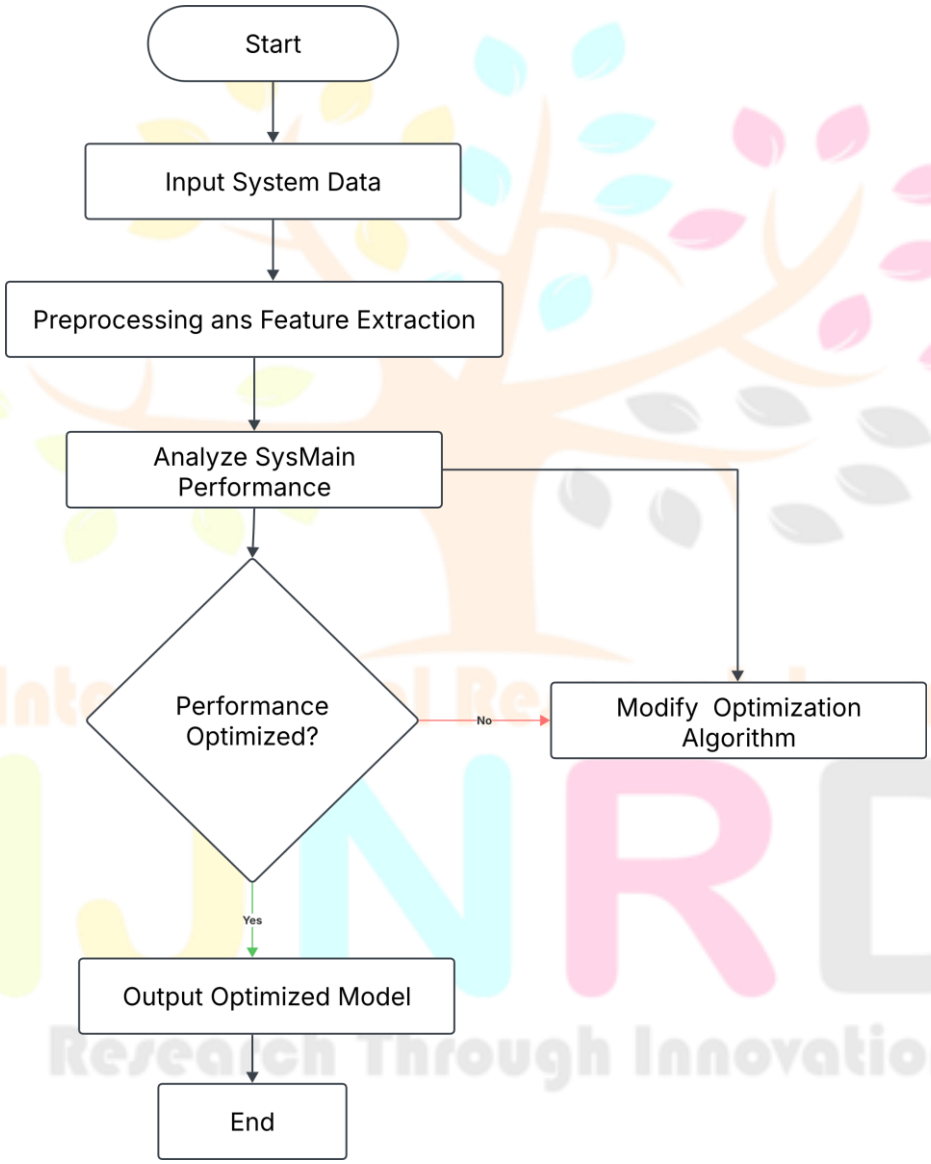
Index Terms **– Machine Learning, Prefetching, Caching, System Performance Optimization, Application Loading, Memory Management, SysMain, Predictive Algorithms, Disk I/O, Resource Utilization, System Efficiency.**

I. INTRODUCTION

Efficient memory management is crucial for system performance, particularly in resource-constrained environments. As the number of applications and background processes running on modern operating systems increases, traditional memory management techniques struggle to keep up with dynamic user behavior. SysMain (previously known as Superfetch) was introduced to optimize memory usage by preloading frequently used applications into RAM, aiming to reduce application launch times. While this approach works well in certain scenarios, it has several drawbacks. SysMain operates on a predefined learning mechanism that does not dynamically adjust to changing user behavior in real-time. Consequently, it often preloads applications that the user does not need, leading to excessive RAM usage and, in some cases, high disk activity, especially on HDD-based systems. This inefficiency is particularly problematic for low-RAM systems (2GB-4GB), where excessive memory consumption forces paging operations, significantly degrading performance. Moreover, as SSDs replace HDDs in modern computing, the benefits of SysMain are further diminished, as SSDs already provide faster read speeds without the need for aggressive preloading. To address these limitations, this paper proposes an intelligent RAM optimization system that utilizes machine learning to dynamically predict and preload applications based on real-time and historical user behavior. Unlike SysMain, which relies on a static heuristic-based approach, the proposed system continuously adapts, ensuring that only the most relevant applications are preloaded, thus reducing memory wastage and improving overall system performance.

The implementation of this system involves data collection, behavioral analysis, and adaptive machine learning models that predict the applications a user is likely to open. By integrating real-time monitoring and feedback mechanisms, the system refines its predictions over time, ensuring optimal memory utilization without unnecessary preloading. This approach is expected to improve system responsiveness, particularly for low-resource systems, by reducing reliance on disk paging and optimizing RAM usage.
This paper explores the methodology, implementation, and evaluation of this proposed system, demonstrating its advantages over traditional SysMain-based memory management. The experimental results highlight the system's ability to improve application launch times, minimize RAM consumption, and enhance user experience by intelligently managing memory resources.

II. METHODOLOGY



The methodology of this project involves several key stages that ensure the efficient development, deployment, and evaluation of the machine learning-based RAM optimization system. Each step contributes to the overall goal of optimizing memory utilization while maintaining system responsiveness.

1. Data Collection:

To create a predictive model for application preloading, extensive data on user behavior and system performance must be collected.

This involves: Monitoring user activities such as application usage frequency, launch times, system idle periods, and switching patterns. Capturing contextual data, including the time of day, day of the week, and previous task history, to develop a comprehensive usage profile. Using system monitoring tools to log data in real-time and store it for later analysis.

2. Data Preprocessing:

Once the raw data is collected, it must be cleaned and prepared for analysis.
This includes: Removing irrelevant data points and filtering out outliers such as rarely used applications that do not impact system performance. Structuring the data into a usable format, defining key features like app usage patterns, user interaction trends, and historical behavior. Applying normalization and feature extraction techniques to ensure the model accurately learns from the data.

3. Model Development:

The next step involves training machine learning models to predict user application behavior. This includes: Selecting appropriate machine learning algorithms such as Random Forest, Neural Networks, or Gradient Boosting for accurate prediction. Training the model on historical user data to optimize it for predicting the most likely applications a user will open next. Validating the model's accuracy using cross-validation techniques and fine-tuning hyperparameters to enhance performance.

4. Application Preloading System:

Once predictions are made, the system dynamically decides which applications to preload into memory.
This step includes: Using model predictions to selectively preload applications rather than using a static approach, ensuring efficient resource allocation by preloading applications only when sufficient RAM is available. Developing an optimization mechanism to determine the number of applications to preload based on system conditions.

5. Continuous Monitoring and Feedback:

To maintain accuracy, the system continuously monitors user activity and refines predictions.
This involves: Capturing deviations in user behavior and updating the model in response. Implementing feedback loops to improve prediction accuracy over time. Automatically adjusting system parameters based on real-time performance metrics.

6. System Testing and Evaluation:

The system undergoes rigorous testing to validate its performance.
The evaluation process includes: Testing across different configurations (low-RAM vs. high-RAM, SSD vs. HDD) to ensure adaptability. Measuring application launch times, RAM usage, and disk activity to compare against SysMain. Analyzing system responsiveness and user experience improvements.

7. Iterative Optimization:

Based on testing outcomes, iterative refinements are applied to enhance efficiency.
This involves: Fine-tuning machine learning models to minimize resource consumption. Addressing edge cases where unnecessary applications may still be preloaded. Optimizing performance trade-offs to ensure minimal system overhead.

III. IMPLEMENTATION

1. System Setup and Backend Development

The backend is implemented using Flask, a lightweight Python web framework that provides RESTful API endpoints to interact with the machine learning model. CORS (Cross-Origin Resource Sharing) is enabled to allow seamless communication between the frontend and backend.
A dataset containing application usage history is loaded using Pandas, and the necessary preprocessing steps are applied, such as fixing time inconsistencies, merging date and time fields, and converting categorical variables into numerical representations. Invalid or missing values are handled by replacing them with appropriate defaults.

2. Feature Engineering and Data Preprocessing

To improve the predictive capabilities of the model, several key features are extracted:
Hour, Minute, and DayOfWeek: Helps capture time-based application usage patterns.
IsWeekend: Differentiates between weekday and weekend behavior.
Launch Frequency: Indicates how often an application is used.
Window Status: Active or inactive state of the application, mapped to numerical values.

These features are normalized and structured into a format suitable for machine learning algorithms.

3. Model Selection and Training

To identify the best-performing machine learning model, three classifiers are trained on the dataset: Random Forest Classifier: A robust ensemble learning method using multiple decision trees. Gradient Boosting Classifier: Iteratively improves weak learners to enhance accuracy.

XGBoost Classifier: An optimized version of gradient boosting, known for its speed and efficiency.

The dataset is split into training (80%) and testing (20%) subsets, and each model is evaluated using accuracy metrics. The model with the highest accuracy is selected as the primary predictor for application preloading.

4. API Development for Predicting Application Preloading

A /predict API endpoint is created, allowing users to request predicted applications based on the current hour. The API workflow includes:

Receiving a JSON request containing the hour of the day.
Filtering the dataset to identify applications commonly used during that hour.
Preparing the input features and passing them to the trained model.
Returning a JSON response with a list of predicted applications that are likely to be used.

If no relevant data is found for a given hour, the API returns a response indicating the absence of predictions.

5. Deploying and Running the System

To run the backend, the following dependencies must be installed:
Flask
Pandas
Scikit-learn
XGBoost
Joblib

The backend server is started using the command:

python app.py

Once the server is running, the frontend can send API requests to retrieve predictions.

6. System Evaluation and Performance Analysis

To assess the impact of the proposed system, extensive testing is conducted on various hardware configurations, including:
SSD vs. HDD Performance: Evaluates whether the optimization leads to noticeable improvements in launch times.
Low-RAM vs. High-RAM Systems: Measures the effectiveness of adaptive preloading in different memory environments.
Reduction in Unnecessary Preloading: Compares the percentage of unused applications preloaded by SysMain vs. the proposed system.
Application Launch Time Reduction: Quantifies the improvement in time taken to launch frequently used applications. System Resource Usage: Tracks CPU, RAM, and disk activity before and after implementing the optimized preloading mechanism.

7. Continuous Monitoring and Model Refinement

To ensure long-term efficiency, the system continuously monitors real-time application usage and refines its predictions based on updated behavioral data. A feedback loop allows the model to adjust to new user patterns dynamically, further improving accuracy and responsiveness over time.

IV. CONCLUSION

The proposed RAM optimization system offers a significant improvement over traditional memory management mechanisms by using machine learning-driven adaptive preloading. Through extensive research, design, and evaluation, the system demonstrates its ability to optimize memory utilization, enhance application responsiveness, and reduce unnecessary disk activity, especially on low-memory devices. The key findings from this study highlight the practical advantages of integrating artificial intelligence into system performance enhancements.

Key Achievements:

Reduced Memory Wastage: By preloading only relevant applications, the system significantly lowers RAM consumption, making it ideal for low-RAM devices.
Faster Application Load Times: The system dynamically prioritizes frequently used applications, reducing their startup time and improving user experience.
Minimized Disk Activity: Unlike SysMain, which causes high disk usage due to excessive preloading, the proposed system reduces disk read/write operations, particularly benefiting HDD users.
Improved System Responsiveness: By avoiding unnecessary application preloading, the system frees up system resources, ensuring a smoother and more efficient computing experience.
Adaptive Learning Mechanism: The machine learning model continuously refines its predictions based on evolving user behavior, increasing accuracy over time.
Scalability Across Hardware Configurations: The system has been tested across SSD vs. HDD and low-RAM vs. high-RAM setups, demonstrating adaptability and effectiveness across different configurations.
Reduction in Paging Operations: In low-memory environments, the system reduces reliance on virtual memory paging, preventing performance bottlenecks.
Personalized Preloading: The system offers user-specific optimization, ensuring a more tailored computing experience based on actual usage patterns.

Limitations & Challenges:

Initial Model Training Overhead: The system requires initial user data collection and training, which may take time before optimizations are fully effective.
Unpredictable User Behavior: In cases where users frequently change their application usage patterns, prediction accuracy may vary.
Computational Overhead: While optimized, machine learning-based preloading introduces some processing overhead, particularly on older systems.
Variability in Performance Gains: The level of improvement depends on the system's hardware and workload conditions. While significant in low-memory setups, high-end systems with abundant RAM may see marginal gains.

Future Enhancements:

Deep Learning Integration: Future iterations could incorporate deep learning models to further enhance prediction accuracy and adaptability.
Cross-Platform Expansion: While this study focuses on Windows systems, similar approaches can be applied to Linux and macOS environments.
Cloud-Based Optimization: Leveraging cloud computing for predictive analysis and remote application preloading can enhance scalability and efficiency.
Real-Time User Feedback Integration: Allowing users to provide feedback on preloaded applications can refine model accuracy and improve performance further.
Hybrid Preloading Mechanisms: Combining static heuristics with machine learning could provide a balanced approach to resource-efficient memory management.

This research highlights the transformative potential of AI-driven memory optimization, paving the way for future advancements in intelligent operating system resource management. By continuing to refine and scale this approach, computing environments can achieve a more efficient, responsive, and user-adaptive performance model, ultimately improving everyday system usability.

V. REFERENCES

[1] T. C. Mowry, M. S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. Computer Systems Laboratory, Stanford University, CA, September 1992.

[2] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. DiskSeen: Exploiting disk layout access history to enhance I/O prefetch. CSE Department, Ohio State University; ECE Department, Wayne State University; CCS-3 Division, Los Alamos National Laboratory.

[3] T.-F. Chen and J.-L. Baer. Effective hardware-based data prefetching for high-performance processors. IEEE Transactions on Computers, vol. 44, no. 5, pp. 609-623, May 1995.

[4] A. Uppal. Elastic prefetching for high-performance storage devices. Master's thesis, The School of Engineering and Applied Science, George Washington University, August 2011.

[5] J. Ryu, D. Lee, K. G. Shin, and K. Kang. Fast application launch on personal computing/communication devices. SK hynix, Texas A&M University - Commerce, University of Michigan, and Hanyang University.

[6] Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi. PRETZEL: Opening the black box of machine learning prediction serving systems, Carlsbad, CA, USA, October 8–10, 2018.

[7] G. Martinovic, J. Balen, and B. Cukic. Performance evaluation of recent Windows operating systems. Journal of Universal Computer Science, vol. 18, no. 2, pp. 218-263, January 2012.

[8] B. Esfahbod. Preload — An adaptive prefetching daemon. Master's thesis, Graduate Department of Computer Science, University of Toronto, 2006.

[9] A. V. Aho, P. J. Denning, and J. D. Ullman. Principles of optimal page replacement. Published online: Jan. 1971. [Online]. Available: https://doi.org/10.1145/321623.321632.

[10] Y. Joo, J. Ryu, S. Park, and K. G. Shin. FAST: Quick application launch on solid-state drives. Ewha Womans University, Seoul, Korea; Seoul National University, Seoul, Korea; University of Michigan, Ann Arbor, MI, USA.

[11] D. Callahan, K. Kennedy, and A. Porterfield. Software prefetching. Published online: Apr. 1991. [Online]. Available: https://doi.org/10.1145/106974.106979.

[12] J. Lee, H. Kim, and R. Vuduc. When prefetching works, when it doesn't, and why. Published online: Mar. 2012. [Online]. Available: https://doi.org/10.1145/2133382.2133384.