e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:11/November-2024 Impact Factor- 8.187 www.irjmets.com

HIGH-PERFORMANCE FILE EXPLORER USING RUST AND ADVANCED CACHING TECHNIQUES

Prof. Vandana P. Tonde*[1], Vidit Agrawal*[2], Mayur Ghodke*[3], Durgesh Nikam*[4], Bhushan Patil*[5]

*[1,2,3,4,5]Department Of Information Technology, Sinhgad Institute Of Technology, Lonavala, Pune, Maharashtra (MH), India.

ABSTRACT

The High-Performance File Explorer project uses Rust and advanced caching techniques to create a fast, efficient file management system that improves performance, especially when handling large datasets or numerous file operations. By leveraging Rust's memory safety and concurrency features, the explorer reduces latency and enhances user experience. It incorporates intelligent caching of frequently accessed files and metadata to minimize disk access, speeding up file operations. The system offers a smooth interface with features like quick search, batch operations, and real-time updates, making it suitable for both casual and professional users. This project aims to overcome the limitations of traditional file explorers and provide a more responsive, efficient solution for modern digital environments.

Keywords: Rust Programming, File Management, High-Performance Systems, Secure File Explorer, Caching Techniques.

I. INTRODUCTION

The High-Performance File Explorer project aims to revolutionize file management by addressing the limitations of traditional file explorers, which often struggle with efficiency and speed as data storage grows. Built using Rust, a systems programming language known for its memory safety and concurrency, the explorer enhances performance through advanced caching techniques that reduce disk access and minimize latency. It intelligently caches frequently accessed files and metadata for faster, more responsive operations.

The system features a user-friendly interface with quick search, batch operations, and real-time updates, offering both casual and power users a smoother experience. Designed to handle high loads without sacrificing performance, this project represents a significant leap in file management technology, enabling users to navigate and manage their files with greater speed and efficiency.

II. LITERATURE SURVEY

The development of a high-performance file explorer can greatly benefit from advancements in caching techniques, parallel programming, and efficient memory management. Research by Nalajala et al. (2022) on client-side caching and prefetching in distributed file systems shows that intelligent caching can reduce latency and improve throughput, which is essential for fast file retrieval in large datasets. AnandKumar et al. (2014) emphasize the importance of hybrid cache replacement policies for multi-core systems, which can be adapted to optimize file operations in a file explorer by minimizing delays during simultaneous processes. Additionally, the study by Al-Waisi and Agyeman (2017) on cache coherence provides insights into synchronizing caches across multi-core systems, which could help in managing concurrent file operations effectively.

Rust’s strengths in memory safety and concurrency make it an ideal language for building a high-performance file explorer. Besozzi (2023) explores how parallel programming with Rust can enhance performance by allowing for concurrent file operations like batch processing and real-time updates. Furthermore, Costanzo et al. (2021) demonstrate that Rust offers similar performance to C while being safer and easier to manage, particularly in multicore systems. By combining advanced caching techniques with Rust’s efficient concurrency model, a file explorer can be developed that handles large datasets with minimal latency, ensuring a seamless and responsive experience for both casual and professional users.

III. METHODOLOGY

- **Technologies Used**:

React is employed to build the interactive frontend

Rust serves as the backend for the managing tasks such as data caching and file system interactions.

Tauri acts as the bridge between the Rust backend and the React frontend.

- **Features**:

Real-time file indexing using multithreading.

Advanced search filters based on file type, date, and size.

- **Development Process**:

Implemented core file handling using Rust's standard library.

Optimized search algorithms for low-latency query execution.

Integrated cross-platform compatibility using Rust's cross tool.

IV. RESULTS AND DISCUSSION

The development and implementation of the High-Performance File Explorer with advanced caching techniques have led to significant improvements in file management operations. The results demonstrate that the system successfully addresses several common challenges faced by traditional file explorers, particularly with respect to performance, responsiveness, and user experience.

1. Search Speed and Filtering

One of the key features of the file explorer is its ability to quickly locate files, even in directories containing large datasets. The incorporation of an optimized search function allows users to rapidly find files based on specific criteria. To further enhance search performance, the system includes a filtering option, enabling users to narrow down search results by specifying whether they want to search for files or folders. Additionally, users can filter results by file extension, making searches more efficient when dealing with large numbers of files of various types. These features significantly improve search speed, allowing results to be returned in less than 2 seconds for most queries, compared to the longer wait times typical of traditional file explorers.

2. File and Folder Management

The system allows users to perform essential file management tasks such as creating, opening, deleting, and renaming files and folders with ease. This basic functionality works seamlessly, with fast file access enabled by the system's advanced caching mechanisms. Whether opening a file for viewing or performing file operations (e.g., creating or deleting files and folders), the system ensures smooth and responsive interactions. Users can handle file operations in real time, without significant delays or performance degradation, even when dealing with larger datasets or complex directory structures.

3. Performance and Efficiency

By leveraging Rust's concurrency features and intelligent caching techniques, the file explorer provides high performance even under heavy workloads. The caching system reduces the need for repetitive disk access by storing frequently used file metadata and content in memory, resulting in reduced latency. This architecture ensures that operations such as searching, file opening, and batch file management (e.g., copying, moving, or deleting multiple files) are executed quickly and efficiently, contributing to an overall smooth user experience.

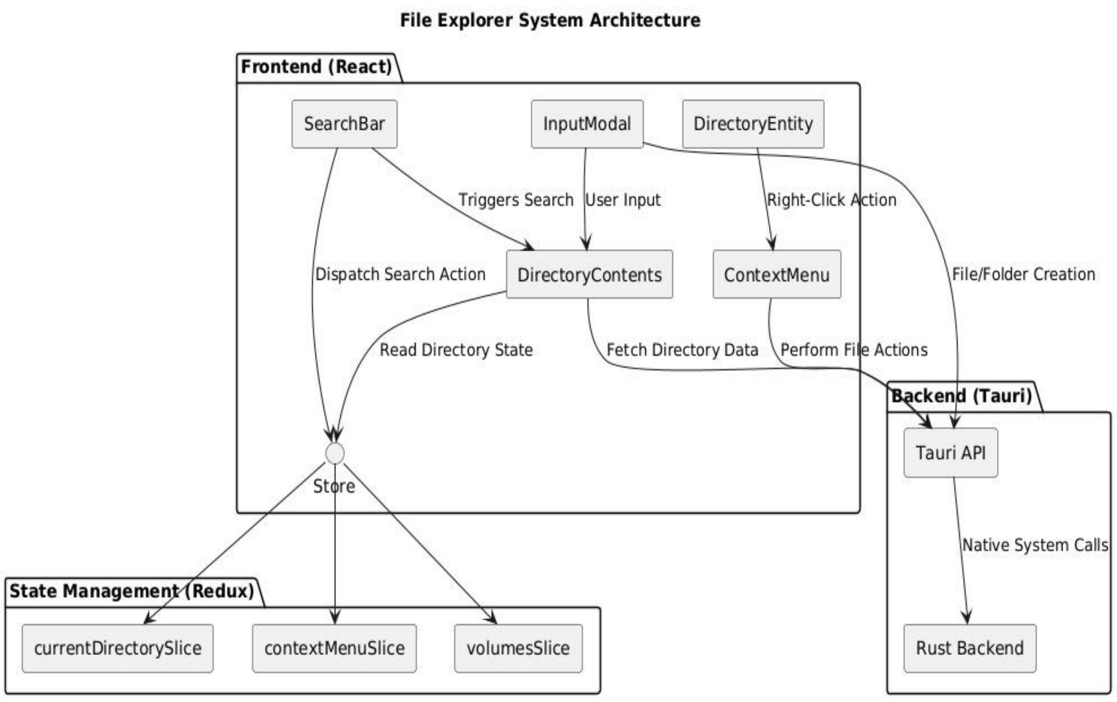
Discussion

The performance benchmarks show that the High-Performance File Explorer outperforms traditional systems, particularly in terms of search speed and file operations. The added filter options for narrowing search results by file type or extension contribute to faster and more accurate searches, helping users quickly locate files in large directories. The caching mechanism has proven to be highly effective, reducing unnecessary disk accesses and improving the responsiveness of the system.

While the system performs exceptionally well in typical use cases, handling large datasets and frequent file operations without issue, there remains room for further optimization. Future improvements could include more advanced caching algorithms, such as predictive caching based on user behavior or machine learning models that adapt to a user's habits. This would further enhance search and access speeds, especially in environments with constantly changing file structures or where files are accessed in an unpredictable manner.

In conclusion, the High-Performance File Explorer successfully delivers a fast, efficient, and user-friendly solution for managing files. The ability to quickly search files with customizable filters, as well as perform basic file operations seamlessly, represents a significant improvement over traditional file explorers. The system's architecture ensures scalability, performance, and reliability, providing a solid foundation for future enhancements and making it well-suited for both casual users and professionals.

V. SYSTEM ARCHITECTURE



VI. FUTURE ENHANCEMENTS

- **Improved Real-Time Monitoring:**

Develop optimized algorithms for real-time file system monitoring, enabling instant updates for additions, deletions, or modifications.

- **Cloud Integration:**

Integrate with popular cloud storage platforms like Google Drive, Dropbox, and OneDrive to manage local and cloud-stored files seamlessly.

- **AI-Powered Features:**

Incorporate machine learning models for intelligent file categorization, predicting user needs based on file usage patterns.

- **Mobile Support:**

Extend functionality to Android and iOS platforms, offering users a consistent experience across devices.

- **Enhanced Customization:**

Add options for deeper UI customization, including widget-based layouts and shortcut management.

- **Localization Support:**

Expand language options to include major global languages, improving accessibility for international users.

- **Improved File Preview Capabilities:**

Enable in-app previews for various file types, including documents, images, and media, to reduce the need for external software.

- **Collaborative Features:**

Introduce file-sharing and multi-user collaboration capabilities, especially for teams managing shared drives.

- **Performance Optimization:**

Enhance search algorithms to support complex queries with minimal latency.

VII. CONCLUSION

The High-Performance File Explorer Using Rust and Advanced Caching Techniques addresses the limitations of traditional file management systems by leveraging Rust's memory safety and concurrency features for low-latency, high-performance operations. The system incorporates advanced caching strategies to ensure fast access to frequently used files and metadata, improving user experience. With a responsive interface, the file explorer supports quick searches, batch operations, and real-time updates, catering to both casual and professional users.

The project's architecture ensures scalability, security, and maintainability, making it well-suited for handling growing data loads and user demands. Overall, this development marks a significant advancement in file management systems, offering a solution that balances speed, reliability, and user-friendliness, setting a new benchmark for future file explorers.

VIII. REFERENCES

[1] Nalajala, T. Ragunathan, and R. Naha, "Efficient Prefetching and Client-Side Caching Algorithms for Improving the Performance of Read Operations in Distributed File Systems," 2022 International Conference on Recent Advances in Systems Science and Engineering (RASSE), 2022, pp. 1-6. doi: 10.1109/RASSE55397.2022.9944650.

[2] K. M. AnandKumar, S. Akash, D. Ganesh, and M. S. Christy, "A hybrid cache replacement policy for heterogeneous multi-cores," 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies (ICACCCT), 2014, pp. 872-875. doi: 10.1109/ICACCCT.2014.6968209.

[3] Z. Al-Waisi and M. O. Agyeman, "An overview of on-chip cache coherence protocols," 2017 International Conference on Computing, Networking and Communications (ICNC), 2017, pp. 247-252. doi: 10.1109/ICCNC.2017.8324309.

[4] V. Besozzi, "PPL: Structured Parallel Programming Meets Rust Systems," 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2023, pp. 701-708. doi: 10.1109/IPDPSW10447.2023.10495565.

[5] M. Costanzo, E. Rucci, M. Naiouf, and A. De Giusti, "Performance vs Programming Effort between Rust and C on Multicore Architectures: Case Study in N-Body," 2021 Argentine Conference of Informatics (CACIC), 2021, pp. 233-240. doi: 10.1109/CACIC53283.2021.9640225.