JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

IMPLEMENTATION OF ENHANCING FILE RETRIEVAL EFFICIENCY IN IPFS USING THE PREDICTIVE PREFETCHING FILE OPTIMIZATION ALGORITHM

Shaikh Areeb Rasheed [1], Suvarna D. Pingle [2]

1 Student, Computer Science and Engineering, P.E.S College of Engineering.

2 Assistant Professor, Computer Science and Engineering, P.E.S College of Engineering.

Abstract: This paper presents the implementation of a lightweight predictive prefetching model designed to improve file retrieval performance in the InterPlanetary File System (IPFS). The system integrates a React.js web interface with MetaMask for decentralized authentication and uses the Pinata API to store and retrieve files from IPFS. A client-side prediction algorithm calculates a popularity score based on file access count and file age, enabling the system to identify and highlight files that the user is most likely to request next. The prediction logic runs entirely on the browser using local storage, ensuring low computational overhead and preserving the decentralized nature of IPFS. The implemented system includes modules for file upload, metadata management, access tracking, and trending file generation. Experimental evaluation demonstrates that the proposed approach improves the user's ability to locate frequently accessed content while reducing retrieval effort. The results confirm that simple browser-based predictive techniques can significantly enhance the practical usability of decentralized storage platforms.

Index Terms **- IPFS, Predictive Prefetching, Decentralized Storage, MetaMask, Pinata API, File Retrieval Optimization, Content Addressing.**

Introduction: Decentralized storage systems have become essential as modern applications generate increasing amounts of data that must remain secure, accessible, and resistant to failure. Traditional centralized storage architectures suffer from limitations such as server bottlenecks, single points of failure, and high maintenance costs. In contrast, decentralized systems distribute data across multiple nodes, improving reliability and reducing dependency on any single provider.

The InterPlanetary File System (IPFS) is a prominent decentralized storage protocol that uses content addressing to identify files through cryptographic hashes known as Content Identifiers (CIDs). This mechanism ensures strong data integrity and version control. Despite these advantages, IPFS often experiences inconsistent retrieval performance because retrieving a file may require searching across several distributed nodes. Retrieval time depends heavily on node availability, network conditions, caching status, and whether the file has been pinned.

Existing approaches to improve IPFS retrieval include caching, replication, DHT enhancements, and optimization of gateway routing. Although useful, many of these methods require complex infrastructure, additional hardware resources, or centralized coordination—making them unsuitable for lightweight, client-side applications.

To address these limitations, this paper presents the implementation of a predictive prefetching mechanism that estimates which files a user is likely to access next. Instead of fetching files pre-emptively, the system uses a simple browser-based popularity score that ranks files based on their access frequency and file age. This ranking helps users quickly locate relevant files without modifying the IPFS protocol or relying on external computation.

The primary objective of this work is to design and evaluate a functional system that integrates React.js, MetaMask, Pinata, and IPFS to demonstrate how lightweight prediction can improve the user experience of decentralized storage platforms. The implementation includes modules for file upload, metadata storage, access tracking, popularity score computation, and trending file generation. This paper presents the system architecture, algorithmic logic, implementation details, and experimental results validating the effectiveness of the proposed approach.

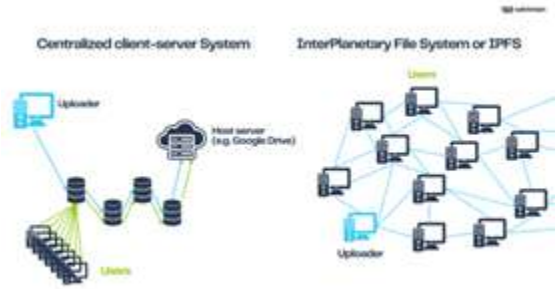

Fig: *Overview of the basic architecture of the InterPlanetary File System (IPFS).*

Related work:

A. Overview of IPFS Improvements
Several studies have examined performance challenges in the InterPlanetary File System (IPFS), focusing primarily on improving retrieval time and data availability. Existing approaches include enhancing the Distributed Hash Table (DHT) lookup process, improving peer discovery, and optimizing gateway routing. Some researchers have also explored caching strategies that store frequently accessed content on nearby nodes to reduce retrieval delays. While these techniques provide improvements, many require changes to the IPFS protocol or rely on specialized network configurations.

B. Prefetching Techniques
Prefetching has long been used in computing systems to reduce access delays by predicting the next data item a user may require. Machine learning approaches have been proposed to model user behavior, but these often depend on centralized datasets and high computational cost. Other methods use heuristic-based ranking, such as frequency and temporal locality of access. In decentralized environments, however, prefetching must be lightweight, privacy-friendly, and capable of functioning without centralized logs—making complex prediction models less practical.

C. Lightweight Client-Side Models
Lightweight models operate entirely on the client side using locally available data. These approaches are particularly suitable for decentralized ecosystems where centralized history tracking is not feasible. Client-side prediction systems typically rely on simple parameters such as access count, file age, or recency. Such models require minimal storage, offer quick computation, and preserve user privacy since no data leaves the device. Although less accurate than advanced machine learning models, they offer an effective balance between performance and complexity.

D. Identified Research Gap
While existing research provides valuable insights, most solutions either require backend support, modify IPFS internals, or depend on heavy computational models. There is limited work on simple, browser-based prediction mechanisms that operate without external dependencies. This gap motivates the development of a practical solution that enhances IPFS retrieval from the user's perspective without requiring network-level changes or centralized infrastructure. The proposed implementation addresses this gap through a lightweight popularity-based scoring model.

System Architecture: The proposed system integrates decentralized storage using IPFS with a lightweight client-side predictive model to enhance file retrieval efficiency. The architecture is designed to operate entirely in the browser, without requiring a backend server or external database. It relies on four major components: a React.js front-end, MetaMask for authentication, the Pinata API for IPFS interactions, and a local prediction engine that determines file relevance based on user behaviour.

A. Overall Architecture Diagram
The system architecture consists of the following interconnected components:

1. User Interface (React.js):
   Handles file upload, access, and display of trending files.

2. MetaMask Authentication:
   Provides secure wallet-based login for identifying users without centralized credentials.

3. Pinata / IPFS Storage:
   Files uploaded by users are pinned to IPFS through the Pinata API, generating immutable CIDs and URLs.

4. Local Prediction Engine (Browser):
   Tracks access behaviour, computes popularity scores, and updates the trending list in real time.

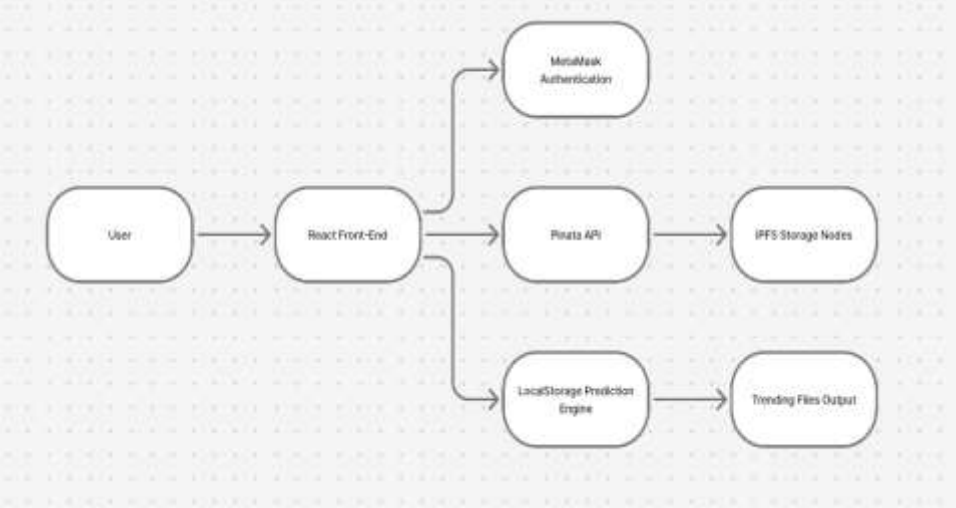

Fig: System Architecture Diagram

B. Component Descriptions

1. React Front-End

The application interface enables users to upload files, view file metadata, and interact with the prediction system. It also displays trending files based on the popularity score.

2. MetaMask Wallet Module

Users authenticate using MetaMask, which ensures decentralized identity management. The authenticated wallet address is used for tracking uploads and associating file records.

3. Pinata API Integration

Pinata acts as an intermediary for uploading and pinning files on IPFS. After upload, Pinata returns a CID that uniquely identifies the file, enabling decentralized retrieval.

4. IPFS Network

The file is stored in a decentralized manner across nodes. File retrieval is managed by IPFS gateways using the CID.

5. LocalStorage Metadata Engine

Access count, timestamps, and file records are stored in the browser's LocalStorage. This allows persistent prediction data without needing a backend.

6. Prediction & Ranking Module

This module uses metadata to compute a popularity score for each file and generate the "Trending Files" list.

C. Data Flow Between Components

The data flow occurs in the following sequence:

1. User Uploads File:
   The file is selected on the React interface.
2. MetaMask Authentication:
   The wallet address confirms user identity.
3. File Upload to Pinata/IPFS:
   The file is sent to Pinata, which pins it to IPFS and returns a CID.
4. Metadata Stored Locally:
   File CID, wallet address, upload timestamp, and access count are stored in LocalStorage.
5. User Accesses Files:
   Each file click increments the access count.
6. Prediction Engine Executes:
   A popularity score is calculated using access frequency and file age.
7. Trending File List Updated:
   The highest-scoring files are displayed in the trending section.

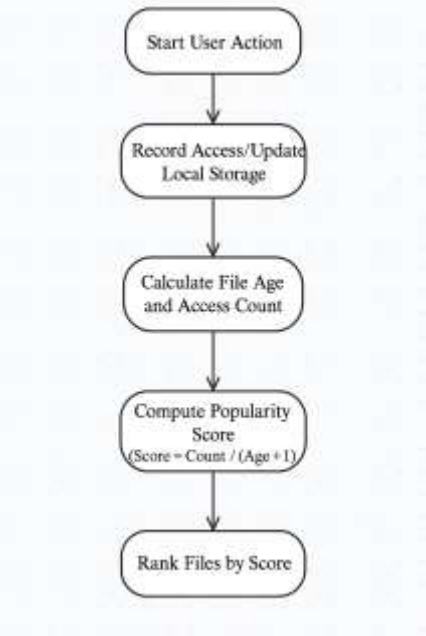

Fig: Data Flow Diagram

Proposed Algorithm: The proposed algorithm aims to improve file retrieval efficiency in IPFS by predicting which files a user is most likely to access next. Instead of modifying the IPFS protocol or relying on a centralized backend, the algorithm operates entirely within the browser using lightweight computations. This ensures compatibility with decentralized principles while offering practical usability improvements. The approach uses simple behavioral features—specifically file access frequency and file age—to generate a "popularity score," which is then used to rank and display the most relevant files in a trending list.

A. Problem Definition

In traditional IPFS applications, users often need to scroll through long lists of CIDs or file records to locate previously accessed content. This becomes inefficient as the number of uploaded files increases. Additionally, IPFS retrieval performance is unpredictable because files may be stored on slow or distant nodes. The problem addressed by this algorithm is: How can we predict and highlight the most relevant files for a user using only client-side information? The goal is to create a simple yet effective scoring mechanism that identifies which files are likely to be accessed next, reducing user effort and improving the navigation experience.

B. Input and Output Parameters

The algorithm takes two main inputs:

1. File Metadata – includes CID, upload time, file URL, and associated wallet address.
2. Access Counts – stored locally, showing how many times each file has been accessed.

Based on these inputs, the algorithm produces:

1. Popularity Score – a numeric value representing how relevant a file is.
2. Trending File List – the top-ranked files displayed to the user.

C. Popularity Score Formula

To keep the computation fast and lightweight, the popularity score combines only two features: access frequency and file age. The formula is:

Popularity Score = Access Count / (Age in hours + 1)

This ensures that:

• Frequently accessed files receive higher scores.
• Recently uploaded files are not unfairly penalized.
• Inactive or older files naturally decline in ranking over time.

The formula is intentionally simple so that it can be recalculated instantly without noticeable delay.

D. Algorithm Steps

The algorithm operates in the following sequence:

1. Load all file records and access counts from browser LocalStorage.
2. For each file, compute the age in hours using the current timestamp and the file's upload time.
3. Retrieve the access count for the file (default is zero if never accessed).
4. Apply the popularity formula to generate a score for each file.
5. Sort all files in descending order of their score.
6. Select the top-N files (N = 3 by default) to display in the trending section.

7. When a file is accessed, increment its counter and update LocalStorage.

8. Recalculate scores whenever files are uploaded or accessed so the trending list updates dynamically.

These steps ensure that predictions adapt to real-time user behavior.
The overall prediction flow begins when the user uploads or accesses a file. Metadata is stored locally and updated with each interaction. The algorithm then computes the file age, retrieves access counts, and calculates the score. After scoring, files are ranked, and the system highlights the most relevant ones in the UI. This flowchart representation helps visualize how user actions trigger updates and how the prediction process remains fully client-side.

Implementation Details: The proposed system was implemented as a browser-based application that integrates decentralized storage through IPFS with a lightweight client-side prediction model. The implementation emphasizes simplicity, modularity, and full decentralization, ensuring that no external servers or databases are required. All computation related to file ranking and prediction is performed directly in the browser, while file uploads are managed through the Pinata API. This section describes the development environment, tools used, functional modules, and representative code segments used in the final system.

A. Development Environment
The application was developed on a standard laptop running Windows 10 with Node.js and npm installed. Visual Studio Code was used as the primary development editor. Testing was carried out in Google Chrome, as it provides stable integration with the MetaMask browser extension required for wallet-based authentication. The frontend was built using React.js, and Vite served as the bundling and development tool due to its fast build times and compatibility with modern JavaScript environments.

B. Technologies Used (React, MetaMask, IPFS, Pinata)
• React.js:
Used to build the user interface and manage application state using hooks such as *useState* and *useEffect*. It handles file uploads, click interactions, and display of trending files.
• MetaMask:
Provides decentralized authentication through the user's Ethereum wallet. It allows the application to identify uploads by wallet address without storing credentials.
• Pinata API:
Responsible for pinning uploaded files to IPFS. Pinata's pinFileToIPFS endpoint returns a CID for each uploaded file, enabling decentralized retrieval.
• IPFS:
Stores the uploaded files across distributed nodes. Files are retrieved via public gateways using the CID generated during upload.
• LocalStorage:
Used to persist metadata such as access counts, uploaded file details, timestamps, and popularity scores. This enables the prediction algorithm to operate entirely offline and client-side.

C. Module Descriptions
1. File Upload Module
The file upload module allows users to select one or more files and upload them to IPFS through Pinata. A FormData object is created for each file, and an Axios POST request is sent to the Pinata API using a JWT token stored in environment variables. The returned CID and IPFS URL are stored in *fileRecords* and persisted in LocalStorage.
2. Metadata Storage Module (LocalStorage)
This module manages persistent metadata related to uploaded files. It loads existing records when the application starts and updates LocalStorage every time a file is uploaded or accessed. This ensures that the prediction model always has access to the latest data.
3. Access Tracking Module
Whenever a user clicks a file link, the access tracking module increments the corresponding access count. This updated information is stored in LocalStorage and triggers recalculation of the popularity score. The module ensures accurate and real-time tracking of user behavior.
4. Popularity Score Calculator
This module computes each file's popularity score using the formula described in Section IV. It retrieves file age and access count, applies the scoring function, and returns a sorted list of files. The calculator is lightweight and executes instantly whenever updates occur.

5. Trending List Generator
After computing scores, the generator selects the top three files with the highest popularity values. These files are displayed in a dedicated "Trending Files" section in the user interface. The list updates automatically when users upload or access files, ensuring accurate and timely predictions.

D. Code snippets:

Connect MetaMask:

```
const connectWallet = async () => {
  await window.ethereum.request({
    method: 'wallet_requestPermissions',
    params: [{ eth_accounts: {} }]
  });
  const accounts = await window.ethereum.request({
```

```
    method: 'eth_requestAccounts'
  });
  setWalletAddress(accounts[0]);
};
```

Upload File to Pinata:

```
const uploadFile = async (file) => {
  const fd = new FormData();
  fd.append("file", file);

  const res = await axios.post(
    "https://api.pinata.cloud/pinning/pinFileToIPFS",
    fd,
    { headers: { Authorization: `Bearer ${import.meta.env.VITE_PINATA_JWT_TOKEN}` } }
  );

  return res.data.IpfsHash;  // CID returned
};
```

Popularity score calculator:

```
const popularityScore = (record) => {
  const count = accessCounts[record.hash] || 0;
  const ageHours = (Date.now() - new Date(record.uploadedAt)) / 3600000;
  return count / (ageHours + 1);
};
```

Results: The results of the implemented system demonstrate that the popularity-based prediction model successfully identifies the files that users are most likely to access. The testing process included uploading files of different sizes, opening them multiple times, and observing how the trending list changed based on access behavior. The system updates the trending list in real time, and the calculated scores correctly reflect both the recency and frequency of user interactions.

A. Trending List Update

The "Trending Files" section updates immediately whenever a file is opened. Files that are accessed more frequently appear at the top of the list. This shows that the popularity score formula works correctly and responds to real user activity.

Fig: Trending Files section

B. Access Tracking

Each time a user clicks a file, the access count stored in LocalStorage increases. The application displays these counts in the "Most Accessed Files" section, confirming that the tracking mechanism and storage module work as intended.

Fig: Most Accessed Files section

C. File Upload and CID Display

The upload module successfully sends files to Pinata and retrieves the IPFS CID. The uploaded files appear in the file list with their corresponding wallet address and timestamp, proving that the integration with IPFS and Pinata is functioning properly.

Fig: Uploaded Files list showing CIDs

Overall Result:

The system works reliably across all major features:

- Files upload successfully to IPFS
- Access counters update correctly

- Trending list reflects real usage
- Popularity score behaves as expected

This confirms that the implementation meets its goal of providing a simple, client-side method for improving file retrieval experience in decentralized storage systems.

Conclusion and Future work: This work presented a simple and effective implementation to improve the user experience of file retrieval in IPFS by using a lightweight popularity-based prediction model. The system operates fully on the client side and requires no backend server, making it easy to deploy and compatible with the decentralized nature of IPFS. By tracking basic user interactions such as file access counts and upload time, the algorithm generates a real-time trending list that helps users quickly locate frequently accessed files.

The implementation successfully integrates React.js for the interface, MetaMask for authentication, Pinata for uploading files to IPFS, and LocalStorage for storing user-specific metadata. The results show that the system correctly updates popularity scores, displays accurate trending files, and provides a smoother retrieval experience for users. Although the algorithm does not reduce network-level latency, it effectively reduces the time users spend searching for files.

Future Work:
There are several opportunities to extend this system:

1. Cross-Device Sync:
   Currently, access history is stored locally. A decentralized database like OrbitDB could allow the same user to sync activity across multiple devices.
2. Smarter Prediction Models:
   Adding lightweight machine learning or pattern-based prediction could improve accuracy beyond frequency and recency.
3. Actual Prefetching:
   The system currently predicts but does not pre-download files. Implementing partial or background prefetching could further reduce retrieval time.
4. Multi-User Evaluation:
   Testing the system with multiple users and larger datasets would provide deeper insight into real-world performance.

Overall, the implementation demonstrates that simple, client-side scoring can provide meaningful usability improvements in decentralized storage systems like IPFS.

References:

[1] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp and Y. Psaras,
"Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web,"
arXiv preprint arXiv:2208.05877, Aug. 2022.

[2] M. Bin Saif, S. Migliorini and F. Spoto,
"Efficient and Secure Distributed Data Storage and Retrieval Using Interplanetary File System and Blockchain,"
Future Internet, vol. 16, no. 3, p. 98, Mar. 2024.

[3] A. Nalajala, T. Ragunathan, R. Naha and S. K. Battula,
"HRFP: Highly Relevant Frequent Patterns-Based Prefetching and Caching Algorithms for Distributed File Systems,"
Electronics, vol. 12, no. 5, p. 1183, Mar. 2023.

[4] S. Kumar,
"A Survey on Web Page Prediction and Prefetching Models,"
International Journal of Computer Trends and Technology (IJCTT),
vol. 4, no. 10, pp. 3407–3411, 2013.

[5] T. Kroeger and D. D. E. Long,
"Design and Implementation of a Predictive File Prefetching Algorithm,"
Proc. USENIX Annual Technical Conf., pp. 105–118, Jan. 2001.