WEP Warwick Evans Publishing

International Journal of Computer Science and Information Technology

ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 2, Number 2, Year 2024
DOI: https://doi.org/10.62051/ijcsit.v2n2.16
Journal homepage: https://wepub.org/index.php/IJCSIT/index

IPFS Keyword Retrieval System Based on Merkle DAG Inverted Index

Lin Shi*, Jun Lu, Xu Zhang, Yuan Cao, Jinchuan Kang

School of Software Engineering, Chengdu University of Information Technology, Chengdu 610000, China

*Corresponding Author: 1176335865@qq.com

ABSTRACT

In recent years, IPFS, as an emerging P2P network technology, has been gaining increasing popularity. There have been numerous studies on the issue of cluster retrieval in P2P networks. This paper proposes an inverted index structure implemented through Merkle DAG, and ensures the eventual consistency of inverted index information among multiple nodes in the cluster through CRDT technology, thereby realizing a relatively effective cluster keyword retrieval scheme based on IPFS. The redundant data is reduced by improving the storage mechanism of cache. The experiments on real data sets prove the effectiveness of the proposed method.

KEYWORDS

IPFS; Merkle DAG; Distributed storage; CRDT; Inverted index

1. INTRODUCTION

Current solutions for content retrieval in IPFS clusters can be categorized into the following two types based on their structural design characteristics:

(1) Centralized Structural Design for Extraction of Retrieval Process In such solutions[1], designers extract the retrieval process and separate it from the operation of IPFS. For instance, in the design scheme proposed by Mao Zheng[2] from Beijing University of Posts and Telecommunications, a centralized streaming media server is used to establish the video streaming workflow. In the design scheme by Shi Linfei[3] and others, the retrieval information API and management tasks are encapsulated within an ES Elasticsearch server[4]. These approaches set up central servers to manage the distribution of data or the maintenance of retrieval information, thereby enhancing the efficiency of retrieval.

(2) Distributed Hierarchical Structure Design with Separation of Keyword Information from File Information Compared to the above centralized structural designs[5], this type of solution features a more fine-grained extraction of functionality, focusing on the generation and dissemination of keyword information for retrieval. For example, Shi Qiue[6] and others from the Chinese Academy of Sciences use TF-IDF and machine learning techniques to generate accurate sentence summaries for uploaded files. The reduced-dimension vectors are then disseminated among neighboring nodes, enabling fuzzy keyword queries and a degree of decentralization.

Combining the advantages of the aforementioned different solutions, this chapter proposes a keyword index storage structure suitable for the IPFS routing mechanism. Upon file upload, a keyword extraction operation is performed on the file contents. The results are converted into fixed-length hash values that are then paired with file content identifiers to form key-value pairs, building the required

keyword structure. In terms of network structure, high-performance nodes are used as full index nodes for different clusters. The keyword index is recorded simultaneously in the full index nodes within the cluster and in the neighboring nodes of the uploading node, enhancing retrieval efficiency both within and between clusters.

2. KEYWORD RETRIEVAL MECHANISM

This paper designs a keyword indexing storage structure suitable for the IPFS routing mechanism. After a file is uploaded, a keyword extraction operation is performed on the content of the file[7]. The result is converted into a fixed-length hash value which, along with the file content identifier, forms a key-value pair, creating the desired keyword structure. In terms of the network architecture, nodes with better performance are used as full index nodes for different clusters. The keyword indexes, when distributed, are recorded both in the full index nodes within the cluster and in nodes near the uploader, enhancing the search efficiency both within and between clusters.

2.1. TF-IDF Algorithm

In practical use, a file management system encompasses more than just text files. When the file type of an educational resource is a text file, the TF-IDF algorithm is used to extract keywords from its content. If the educational resource file type is a video, image, or another type that makes it difficult to extract text information, the TF-IDF algorithm performs keyword extraction operations on the file name, or keywords can be defined by the user. The TF-IDF algorithm can extract keywords of educational resource files, with its core principle being to represent the file with one or several keywords that have feature weight values exceeding a threshold[8]. In this system, the maximum number of keywords is set to five. TF-IDF uses Term Frequency TF to indicate how frequently a keyword appears in a file. The more frequently it occurs, the higher the TF value, making it more representative of the educational resource file.

$$TF_{i,d} = \frac{f_{i,d}}{\sum_{j=1}^{n} f_{j,d}} \tag{1}$$

In the formula, $f_{i,d}$ represents the frequency at which the keyword appears in document *d*, and $\sum_{j=1}^{n} f_{j,d}$ is the total number of occurrences of all keywords in document *d*.

The aforementioned TF calculation part is about extracting high-frequency words that carry meaning, aside from common stop words and other irrelevant terms from the document. Meanwhile, the inverse document frequency IDF calculation part within TF-IDF serves to rank these high-frequency and meaningful words by importance

$$IDF_i = log \frac{D}{d_i + 1} \tag{2}$$

In the formula, *D* represents the total number of documents in the corpus, k_i denotes the number of documents containing the k_i. Adding one to the denominator is to prevent the situation where the keyword does not appear in the corpus at all, which would make d_i equal to zero. The IDF function determines the k_i's importance in categorizing documents based on the number of appearances of k_i within the corpus. The less frequently k_i appears across numerous documents, the higher the distinction level of k_i.

$$W_{i,j} = TF_{i,j} * IDF_i \tag{3}$$

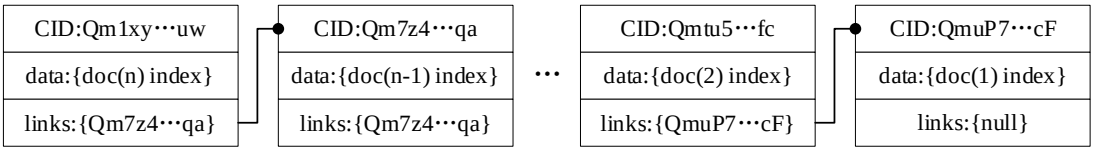
$W_{i,j}$ represents the weight value of the k_i. The core concept of the TF-IDF algorithm is to consider both the importance of keywords within a document and their prevalence across documents. By integrating these two parameters, the algorithm identifies and selects a few keywords that are most valuable for summarizing and distinguishing the document

2.2. Inverted Index Structure and Update Strategy based on Merkle DAG

The universal data structure of IPFS is Merkle DAG, which is a tree-like structure[9]. Through the links between nodes, it can construct any structured and unstructured data. IPFS allows developers to manipulate Merkle DAG objects. Therefore, nodes within the structure can be manually set. Since updating the Merkle DAG requires recalculating hash values, there are different update costs for different Merkle DAG structures. There are three specific manifestations of Merkle DAG: flat structure, balanced structure, and linked structure.

The linked structure, due to its structural characteristics, allows for the option to add new elements either at the head or the tail. The difference between the head insertion method and the updates of the aforementioned structures lies in the fact that the new node itself becomes the root node. Therefore, only the new node itself needs to be hashed, resulting in the lowest and fixed computational cost. On the other hand, the computational cost of tail insertion is the same as that of the balanced structure.

In summary, adopting a linked structure as the implementation of an inverted index minimizes update costs. Therefore, the system designed in this paper employs a singly linked structure as the storage structure for the Merkle DAG-based inverted index table and uses the head insertion method as the update strategy. The singly linked structure is illustrated in Figure 1.



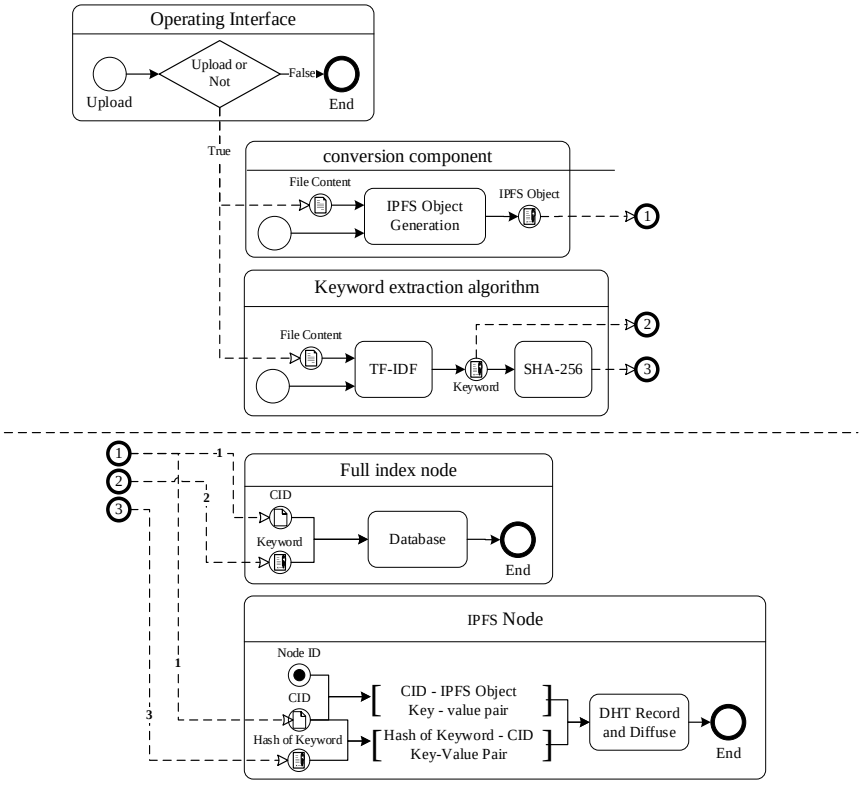
1**Figure 1.** Inverted index single chain structure diagram

The specific update process for the head insertion method involves the following steps: Before each update, a reference to the original root node is created. Then, a new root node containing the file information is established and linked to the previous root node. Finally, the data is persisted on IPFS, and a new content identifier is returned.

2.3. Index Building and Distribution Process

In the native IPFS technology, the file upload process includes the file content segmentation and the Merkle DAG object generation workflow[10], and ultimately returns the file's Content Identifier CID. The Key-Value structure, consisting of the CID and the node identity, is the main content of the DHT DistributedHashTable. To implement a name-based retrieval function that supports keyword searches, the KS-HFSS system designed in this paper conducts keyword extraction concurrently with the file conversion process.

The process of publishing index information essentially involves distributing the index data to other nodes in the cluster. In IPFS, the distribution of index information is traceable. Since both the file content identifier and the node ID are hash values, the native IPFS file content identifier distribution process involves performing an XOR operation between the CID and node ID, with the resulting value representing the logical distance. As the file CID propagates through the node routing, it will gradually find a fitting node to store the corresponding record, a node whose ID value has the shortest logical distance from the CID. Therefore, to utilize this feature of IPFS, the result of hashing the keyword is paired with the CID to form an index, which is then published to a suitable node using IPFS commands. Concurrently, the keyword index is sent to all index nodes within the cluster, forming an inverted index table.The workflow is shown in Figure 2.



2**Figure 2.** Inverted index single chain structure diagram

The inverted index table on the full-index node is maintained by a storage engine, and the mapping relationship between words and CIDs is relatively straightforward. For scenarios involving large amounts of data storage and retrieval, non-relational databases perform more efficiently, therefore the system uses a non-relational database to maintain the inverted index table. Since there are multiple full-index nodes, and given that in practical applications, compared to general storage systems, there is a higher proportion of single-point write operations; moreover, as each node in the IPFS system can handle part of the read operations, the system adopts LevelDB to persist index information. LevelDB, together with the Merkle DAG-based inverted index structure, constitute the keyword search service on the full-index node. The complete process of index construction is illustrated in Figure 3-3, where the file is uploaded to IPFS, the file content identifier is returned, and the indexing begins. The extracted keywords, text information, and file type are combined to form the metadata of a Merkle DAG node. After the file information is uploaded to the full-index node, the inverted index's singly linked structure corresponding to the keyword is scanned to determine whether it contains the file's CID record and whether there is a need to insert this element into the singly linked structure. If every keyword's singly linked structure already contains the current file index, the process ends. If any keyword's singly linked structure does not contain the current file index, a Merkle DAG node that stores the index entry is created. This node is then linked to the head of the current singly linked structure following the head insertion method. Subsequently, the root node of the singly linked structure is updated, and the new pair is stored in LevelDB. After LevelDB completes updating the mapping relationship, the index-building process is complete.

3. SYSTEM FILE MANAGEMENT AND SYNCHRONIZATION POLICIES

In terms of file backup work, IPFS employs a passive caching method, that is, when one node requests data from another node, data blocks and index information are temporarily cached on the nodes along the route. However, in the application context of this system, this scheme cannot alleviate network congestion and reduce retrieval time in the event of sudden high-concurrency requests. Therefore, to

address the aforementioned issue, a proactive backup management approach should be constructed to facilitate the system's allocation of network resources and file backup. Furthermore, considering the particularities of the application context, a mathematical model should be constructed based on factors such as the cluster node's online rate, node reliability, predicted file popularity values, and the cost of file backups, to manage the cache area scientifically.

3.1. Algorithm for Controlling the Number of File Copies

3.1.1. File Backup Cost Calculation

The success of file replication is related to the effectiveness of the target node. Given that data nodes in the IPFS network are independent of each other, it can be assumed that the reliability of IPFS storage nodes follows an exponential distribution over time *T*.

$$E(T) = e^{-\lambda T} \tag{4}$$

Where λ is the node's failure rate. Based on real-world usage conditions, this can be adapted to online rate, thus *E(T)* represents the expected value of node availability within a certain period *T*. Incorporating the file transfer time between nodes, Ttrans, and the number of nodes, the formula for calculating the reliability factor f can be derived.

$$f = E(T)^n = e^{-\lambda T * n} \tag{5}$$

The more file blocks there are, the lower the probability of successful transmission. The average transmission cost of a file, Cave, has a calculation formula.

$$C_{ave} = T_{trans} * (1 - f) \tag{6}$$

3.1.2. Calculate the Number of File Copies

To maximize file retrieval efficiency, it is essential to control the number of data replicas reasonably. Suppose a file, Doc, is divided into multiple data blocks for storage, with each block stored as a unit.

$$Doc = \{Block_1,\ Block_2,\ \cdots,\ Block_n\} \tag{7}$$

In the KS-HFSS system, replicas are stored across multiple nodes. Based on the reliability analysis of nodes, the unreliability probability $P(\overline{Block_x})$ of a data block and its *m* replicas can be expressed mathematically.

$$P(\overline{Block_x}) = \prod_{y=1}^{m}\left(1 - e^{-\lambda_y T_m}\right) \tag{8}$$

thereby deriving the formula for the probability of data block availability.

$$P(Block_x) = 1 - \prod_{y=1}^{m}\left(1 - e^{-\lambda_y T_m}\right) \tag{9}$$

T_m represents the storage life cycle time of the m replicas of $Block_x$ on IPFS nodes. To ensure the availability of a file, all of its data blocks must be available; the unavailability of even a single block would prevent the file, Doc, from being successfully assembled, rendering it unavailable.

In the IPFS storage network, a file consists of several data blocks stored on different nodes, which are independent of one another; hence, it can be inferred that the data blocks are independent. Based on the extended formula of probability theory, the availability rate of file Doc is as follows.

$$P(Doc) = \prod_{x=1}^{n} P(Block_x) = \left(1 - \prod_{y=1}^{m}\left(1 - e^{-\lambda_y T_m}\right)\right)^n \tag{10}$$

Assuming in the KS-HFSS system, users have a certain expectation for retrieving file Doc, then to determine the minimum number of replicas m, the formula can be derived when the file availability rate satisfies formula.

$$P(Doc) = \left(1 - \prod_{y=1}^{m}\left(1 - e^{-\lambda_y T_m}\right)\right)^n \geq E(A) \tag{11}$$

$$m \geq -(log_{e*n} E(A)/\lambda T) \tag{12}$$

By the formula above, the minimum number of replicas m can be deduced. However, in practical use scenarios, other factors should also be considered when determining the number of backups. Therefore, this text introduces the node online rate Nonline, the predicted file popularity Fpred, and the file backup cost Cave, previously calculated, into the formula for calculating the optimal number of replicas m'.

$$m' = \left\lceil m * \left({}^{N_{online} * F_{pred}}\Big/_{C_{ave}} + 1 \right) \right\rceil \tag{13}$$

The formula, based on the known minimum number of replicas m, adjusts the number of replicas based on the node online rate, predicted file popularity, and backup cost to reduce the negative impacts of backup costs while meeting reliability requirements as much as possible. Increasing the number of replicas can enhance reliability but will also raise the backup cost, hence the need for this balance.

3.2. Inverted Index Synchronization of Multiple Nodes

The IPFS system, as a distributed system, follows the CAP principle in terms of file synchronization, meaning that consistency, availability, and partition tolerance cannot be achieved simultaneously. The platform software designed in this paper is a system with numerous nodes and different clusters. Therefore, the design philosophy of this paper is to sacrifice strong consistency in order to ensure the system's availability and partition tolerance. This paper adopts eventual consistency as the requirement for index information consistency among all index nodes across clusters. Achieving eventual consistency requires fulfilling three characteristics: transaction propagation, full convergence, and operation termination. This demands that weak consistency synchronization should ensure that synchronization information can propagate to all nodes, and that nodes can achieve strong data state convergence after a finite number of operations.

3.2.1. Node Communication Based on Message Subscription Publishing Model

Publish-Subscribe, abbreviated as "PubSub," is a pattern commonly used to handle information exchange in large-scale networks. In this pattern, a "publisher" can send messages in different topics, while a "subscriber" only receives messages from topics they are interested in, with no direct contact between them. This method provides greater network scalability and flexibility.

In the system design, nodes can be classified into upstream nodes and downstream nodes based on the production relationship of indexes. Nodes responsible for content analysis, index establishment, and publishing are called upstream nodes, while nodes waiting for index messages in the topics are called downstream nodes. Based on whether they will have an impact on the entire system, node methods can be categorized into Influence-Spreading methods and Self-Adjustment methods. Updating the local index of nodes is a self-adjustment method, while publishing new indexes to the network is an influence-spreading method. After the self-adjustment method of upstream nodes is executed, it triggers the execution of the influence-spreading method. Downstream nodes receive new indexes, perform local updates, and continue to spread. Newly added nodes in the cluster first subscribe to index-related topics, establish connections with guiding nodes, and then broadcast their information in the network to facilitate information exchange among multiple nodes. In large-scale networks, nodes are divided into multiple sparse clusters based on different subscription topics. The data exchange between clusters is achieved through the index synchronization of the full index nodes in each cluster, publishing data states from one cluster to another.

3.2.2. Index Table Merging Mechanism based on G-Set

Taking advantage of the feature that files will not be deleted after being pinned on IPFS nodes, the system adopts G-Set as the data type to synchronize the inverted index entries. G-Set is a type of CvRDT, and its operation principle is as follows table 1.

Table 1. G-Set structure definition table

Augmented G-Set Based on State Changes
// The Definition of G-Set Structure **Struct** G-Set: elements = ∅ // **The Operation of Adding** **function** add(G-Set A, element e): if e ∉A.elements: A.elemenets.add(e) // **The Operation of Query element Inclusion** **function** container(G-Set A, element e) → boolean: **return** e ∉A.elemenets or not // **The Operation of Merge** **function** merge(G-Set A, G-Set B) → G-Set: G-Set C = new G-Set C.elements = A.elements ∪B.elements **return** C

Assume there are two full index nodes, Node A and Node B, both containing a single linked list of inverted indexes corresponding to the keyword K with head node CIDs Hash A and Hash B respectively. When the nodes merge index tables, they perform the merge work based on the comparison of the Merkle DAG node hashes and the merging process of the G-Set structure. The specific workflow is shown in table 2.

Table 2. Chain Index Merging Process

1.	Node A obtains the file index index with the keyword K；
2.	Node A generates a Merkle DAG node c for index and sets c's links to a, resulting in (c, {a}). It then updates the head node of the single linked list structure, changing Hash A to Hash C, and records it in LevelDB；
3.	Node A uses the PubSub component to publish Hash C in the index update topic；
4.	Node B receives the message and establishes an index with Node A. It retrieves the entire linked list content of the inverted indexes using Hash C and searches for nodes not included in its local inverted index for the keyword K, storing them in set S;
5.	If S is empty, it means Node B's inverted index linked list is the same as Node A's, and the operation ends here.
6.	If S is not empty, the records in S are arranged in the order they were obtained and added to Node B's inverted index linked list accordingly.
7.	If Hash B ⊆ Hash C, it means Node B's inverted index for keyword K is a subset of Node A's content. After adding the records, Node B uses Hash C as the new head node CID for its inverted index linked list.
8.	If Hash B ⊈ Hash C and Hash B ⊉ Hash C, it means the states of the inverted index linked lists on both nodes cannot be compared. In this case, no merging is performed. Instead, the hash value of the other node's linked list structure is added to the records for keyword K in the LevelDB of both nodes.

3.3. Experiment and Analysis

3.3.1. Experimental Environment

Table 3. Information of Experimental Environment

System	
Linux version	Ubuntu 22.04 LTS
Go version	1.19.1
Hard disk	500G
Memory	16G
Network bandwidth	10Mbps

3.3.2. Experimental Result

In the IPFS network, the time of data retrieval is influenced by multiple factors, including network topology, the quality of connections between nodes, and the efficiency of data distribution, among others. Because IPFS uses Distributed Hash Tables (DHT) for data retrieval, the actual retrieval time is dynamic. In this study, the experimental environment is as described above, with variables including the number of cluster nodes and data block sizes. The number of nodes varies from 5, 10, 20, to 50, with network bandwidths of 1Mbps and 10Mbps. Data block sizes are 5MB, 10MB, and 20MB. The experimental results are shown as follows.

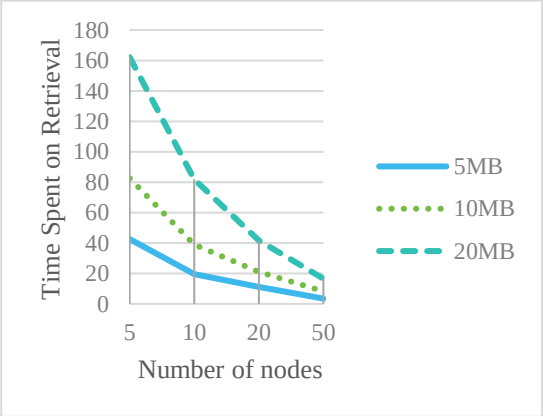


Figure 1. Time Consumption Trend Chart in Traditional IPFS Network

(Network Bandwidth Equal 1Mbps)

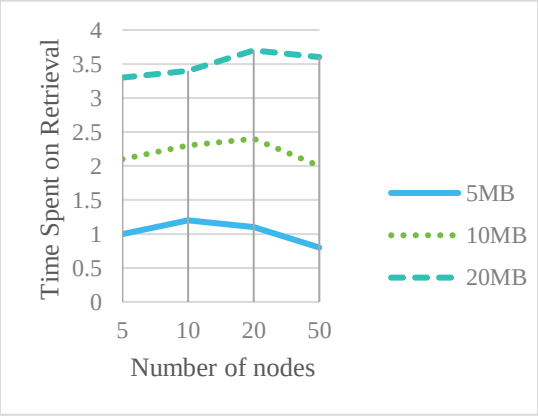


Figure 2. Time Consumption Trend Chart in New IPFS Network

(Network Bandwidth Equal 1Mbps)

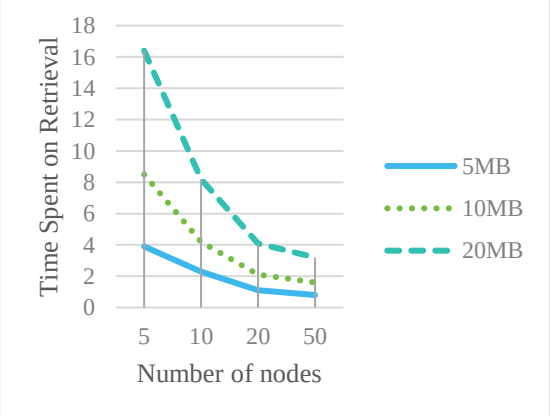


Figure 3. Time Consumption Trend Chart in Traditional IPFS Network

(Network Bandwidth Equal 10Mbps)

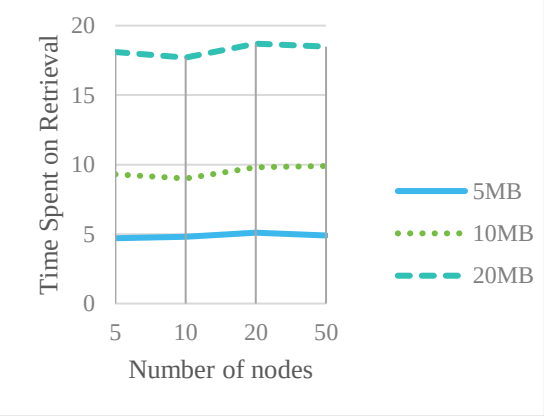


Figure 4. Time Consumption Trend Chart in New IPFS Network

(Network Bandwidth Equal 10Mbps)

Based on the above, it can be seen that in the IPFS cluster structure described in this article, the longer data retrieval time caused by multiple DHT lookups is avoided. The time appears relatively stable (with fluctuations depending on the time it takes for ordinary nodes to establish connections with persistent nodes).

4. CONCLUSION

This paper first briefly discusses the drawbacks of the existing centralized network architecture and the limitations of the IPFS architecture, clearly stating the objectives of the work. Then, it proposes a hierarchical indexing scheme based on Merkle DAG. In this scheme, Merkle DAG is used to maintain the associated information of keywords, file content, and file information. The identifier of this DAG serves as the starting point for traversing all content. When a user obtains the DAG CID where the target file is stored through the top-level index, they can traverse all file metadata in the DAG, thereby achieving the process of users obtaining the complete content of a file through keyword search.

REFERENCES

[1] Kang peng, Yang Wenzhong, Zheng Jiong. Blockchain PRivate File Storage-Sharing Method Based on IPFS[J]. Sensors, 2022, 22(14)

[2] BENET J. IPFS-content addressed, versioned, p2p file system[j]. arXiv E-print, 2014: arXiv: 1407.3561.

[3] Khudhur N, Fujita S. Siva-The PFS search engine[C]/2019 Seventh International Symposium on Computing and Networking (CANDAR). IEEE, 2019: 150-156.

[4] Zhu Liyan, Xiao Chuqiao, Gong Xueqing. Keyword search in decentralized storage systems[J]. Electronics, 2020, 9(12): 2041.

[5] Klems M, Eberhardt J, Tai S, et al. Trustless intermediation in blockchain-based decentralized service marketplaces[C]/International. Conference on Service-Oriented Computing. Malaga: Springer, Cham, 2017: 731-739.

[6] Mosharraf sharafat Ibn Mollah, Adnan Muhammad Abdullah. Improving lookup and query execution performance in distributed Big Data systems using Cuckoo Filter[J]. Journal of Big Data,2022,9(1).

[7] Huang sui, Li Jian, Fan Bingbing. IABC: A cross-domain authentication method based on blockchain and cuckoo filter [J]. Small Microcomputer system, 2020, 41(12): 2620-2625.

[8] Deng yingying. Optimization and Application of Cuckoo Hash Table [D]. Nanjing university of posts and Telecommunications, 2021. DOI: 10.27251/d.cnki.gnjdc.2021.000705.

[9] BRUIN M D. Search engine for the interplanetary file system[EB/OL].[2020-12-21].https：//github.com/ipfs-search/ipfs-search.

[10] KHUDHUR N, FUJITA S. Siva-the IPFS search engine[C]//Proceedings of the 2019 Seventh International Symposium on Computing and Networking(CANDAR), Nagasaki, Japan, Nov 26- 29, 2019.Los Alamitos: IEEE Computer SOC, 2019: 150-156.