ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Optimized Search Solution for Storing and Retrieving Large Files of Legacy Systems

Arjun Reddy Lingala

USA

ABSTRACT

As organizations transition from legacy systems to modern architectures, managing and retrieving historical data efficiently becomes a significant challenge. Traditional relational databases are often unsuitable due to their high storage costs and limited scalability. In addition to this, organizations acquire other organizations and the storage engines used in legacy companies will be slowly migrated to new processes that the parent organization uses, but the companies wanted to retain the data of legacy systems for auditing and analysis purposes. Approach discussed in this paper involves converting structured and unstructured legacy system data into files, which are then stored in HDFS for cost-effective, distributed storage. To enable fast search and retrieval, we employ Elastic Search to index metadata and key terms extracted from these files. Since Elastic Search is designed for real-time indexing and full- text search, it allows users to perform rapid lookups based on predefined attributes. However, storing complete file metadata in Elastic Search can be inefficient. To optimize the process, we leverage HBaseas a NoSQL mapping layer that links search indices to the corresponding HDFS file paths. This ensures that, rather than storing entire file details within Elastic Search, only essential metadata is indexed, and full records can be retrieved efficiently using HBase as a key-value lookup store. The proposed system optimizes both storage costs and query performance by distributing large data across HDFS while leveraging the indexing capabilities of Elastic Search and the fast lookup capabilities of HBase. This architecture is particularly beneficial for enterprises dealing with regulatory compliance, audits, and historical data access, where retaining legacy data is essential but needs to be both cost-effective and easily searchable. The study concludes that the combination of HDFS for distributed storage, HBase for index mapping, and Elastic Search for keyword-based searching provides an optimal balance between cost efficiency and performance for managing legacy system archives.

***Corresponding author**

Arjun Reddy Lingala, USA.

Received: February 05, 2022; **Accepted:** February 10, 2022, **Published:** February 20, 2022

Keywords: Searching, Distributed Systems, Legacy systems, Large files, Distributed systems, Key-Value mapping, Elastic Search, HBase, Search Indexes, NoSQL

Introduction

In today's digital landscape, organizations accumulate vast amounts of data from various systems, many of which even- tually become legacy or deprecated due to technological advancements or system migrations [1-3]. Managing and retrieving historical data from such systems presents several challenges, including high storage costs, inefficient querying mechanisms, and scalability limitations. Traditional relational databases, commonly used in legacy systems, struggle with performance issues when handling large datasets, making retrieval opera- tions slow and resource intensive. As a result, enterprises seek optimized solutions to store, index, and efficiently retrieve data while maintaining cost-effectiveness. To address these chal- lenges, we propose an optimized search solution that integrates Elastic Search, HBase, and Hadoop Distributed File System [2,3]. The core idea behind our approach is to extract data from legacy systems, convert it into large files, and store them in HDFS, which offers a scalable, fault-tolerant, and cost-efficient storage infrastructure [1]. Since querying data directly from HDFS is inefficient due to its distributed nature, we employ Elastic Search to create searchable indexes using key metadata and terms extracted from the files [2]. However, since Elastic Search is primarily designed for search rather than persistent storage, we introduce HBase as a mapping layer to efficiently link the indexed metadata to the corre- sponding file locations in HDFS [1,3]. The proposed system architecture ensures an optimal balance between storage effi- ciency, search performance, and retrieval speed using HDFS for storage, Elastic Search for fast indexing, and HBase as mapping layer [2]. This three-tiered architecture provides an efficient and scalable alternative for handling large volumes of legacy system data. By leveraging HDFS for cost-effective storage, Elastic Search for high-speed indexing, and HBase for optimized data retrieval, our solution significantly improves query response times and reduces the overall computational and storage overhead associated with legacy data management [1].

System Architecture

The architecture is designed to extract, store, and index data from legacy systems while maintaining cost-effectiveness and high-speed search capabilities. The raw data from legacy systems is first converted into files and stored in HDFS, leveraging its distributed, fault-tolerant, and cost-efficient storage model [1]. Since searching for specific data within HDFS is inherently slow due to its file-based structure, Elastic Search is used to create search indexes containing metadata and key terms extracted from the files [2]. However, because Elastic Search is primarily designed for high-speed indexing and search rather than data storage, HBase acts as a mapping layer, linking the indexed records in Elastic Search to their corresponding file paths in HDFS [3]. This structured approach ensures that users can quickly search for and retrieve large files with minimal latency, even when dealing with extensive

datasets. The architecture consists of three core components: Storage layer, Indexing and Search layer, Mapping layer.

Storage

HDFS or other distributed storage platforms like S3 serves as the primary storage repository for all data extracted from legacy systems [1,4]. Given that legacy databases often store information in structured formats in some cases, the data is converted into structured or unstructured files before being stored in HDFS. HDFS provides several advantages like scalability, fault tolerance, and cost effectiveness. HDFS is not optimized for fast querying. Searching for a specific piece of information within a vast collection of files can be computationally expensive, making a dedicated search layer necessary.

Indexing and Searching

Elastic Search is used to address the search inefficiencies of HDFS by providing real-time, full-text search capabilities [1,2]. Instead of searching entire files directly within HDFS, key metadata and search terms are extracted from each file and indexed into Elastic Search. This ensures that when users search for specific terms, the system can quickly return relevant results without scanning entire datasets. Elastic search provides key capabilities like high speed search, metadata indexing, and scalability. Elastic Search is effective at indexing and retrieving search results, but it is not designed to store large files where a key value storage like HBase can be used.

Mapping

HBase is integrated into the system as an intermediary data store, which links Elastic Search index entries to actual file locations in HDFS [2,3]. Since Elastic Search only indexes metadata and key terms, it does not maintain references to the complete dataset. HBase stores a mapping between indexed records in Elastic Search and the corresponding file paths in HDFS, ensuring that relevant data can be retrieved quickly without scanning entire directories. HBase provides key value mappings, optimizes retrieval by eliminating the need for full- directory scans by directly linking indexed terms to stored files and ensures scalability.

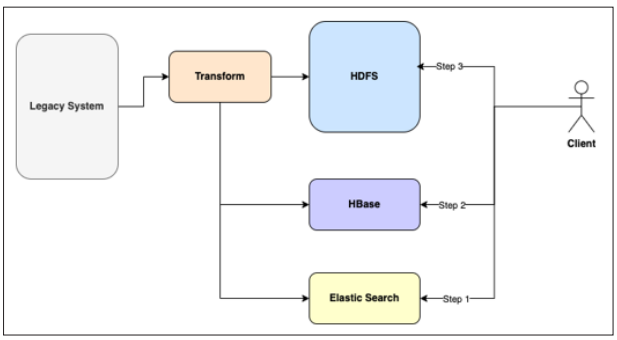

Figure 1: System Architecture and Data Flow

Data Flow and Execution

The efficiency of the proposed search solution is rooted in a well-structured data flow and query execution mechanism. The system follows a structured process to ensure that legacy system data is effectively stored, indexed, and retrieved without unnecessary computational overhead. The data flow consists of two phases which include Storage and Indexing – which involves extracting, storing, and indexing large files in HDFS while capturing relevant metadata in Elastic Search and establishing lookup mappings in HBase Search and Retrieval – handles user search queries, identifying relevant indexed records in Elastic Search, retrieving the corre- sponding HDFS file locations via HBase, and ultimately fetching the requested files from HDFS [1-3].

Storage and Indexing

The first phase of the workflow is dedicated to the systematic extraction, storage, and indexing of legacy system data [5]. It involves converting structured and unstructured data from legacy systems into file formats suitable for scalable storage and optimized retrieval.

Data Extraction from Legacy Systems: Legacy systems often contain large volumes of structured and unstructured data stored in relational databases, flat files, logs, or other proprietary systems. Extracting this data requires careful han- dling to ensure data integrity and format compatibility. Data stored in relational databases is exported using SQL queries, typically formatted into CSV, JSON, or XML files. Data from system logs, application reports, scanned documents, and free- text repositories is extracted in formats such as plain text, PDF, or other readable files. Before storage, the extracted data undergoes cleaning, deduplication ensuring consistency.

Storage: Once the data is converted into appropriate file formats it is stored in HDFS, taking advantage of its fault- tolerant, distributed, and cost-effective storage capabilities. Each file is automatically replicated across multiple nodes to ensure data availability and resilience against failures. Com- pression techniques are applied to reduce storage overhead and improve access speeds.

Indexing: Since searching for specific content directly within HDFS is computationally expensive, an indexing mech- anism is introduced using Elastic Search. The system extracts key metadata and relevant search terms from each stored file and indexes them to enable rapid query execution. Essential attributes such as file name, creation date, file type, originating system, and other key terms from the data file are captured as part of metadata extraction and indexed within elastic search cluster enabling full-text search and quick retrieval. Elastic Search ensures that the indexed data is distributed across multiple nodes, ensuring high availability and low latency querying [2].

Mapping Search Indexes: Elastic Search efficiently in- dexes metadata, it does not store the actual data. To facilitate efficient retrieval, HBase acts as an intermediary mapping layer that links Elastic Search records to the corresponding file locations in HDFS. Each indexed record in Elastic Search

is mapped to its corresponding HDFS file path within HBase. HBase organizes mapping tables in a column-family structure, optimizing lookups for search results [3]. The lookup mechanism is designed to handle large-scale datasets while maintaining high throughput and low latency.

Search and Retrieval

The second phase of the workflow focuses on executing user search queries, retrieving metadata from Elastic Search, identifying file locations via HBase, and fetching complete datasets from HDFS [2].

Query Execution: When a user initiates a search request, the system processes the query using Elastic Search, which rapidly retrieves relevant metadata records. The system ana- lyzes the search query to determine whether it requires exact match, phrase

search, or keyword-based lookup by interpre- tation. Elastic Search scans the indexed metadata and returns the most relevant records in milliseconds. Search results are ranked based on relevance and keyword match strength, with filtering options for refining results.

Retrieve File Location: The next step involves querying HBase to retrieve the corresponding file locations in HDFS. Using the unique document ID returned by Elastic Search, the system performs a direct lookup in HBase to retrieve the associated HDFS file path. HBase's NoSQL architecture allows for high-speed mapping retrieval, eliminating the need for complex relational joins [1].

Fetch File: After obtaining the HDFS file path from HBase, the system proceeds to fetch the actual data from HDFS. Depending on the user's requirement, the system either streams the requested file in chunks for real-time access or provides an option for full file download [3].

Performance Optimization

To ensure the efficient storage, indexing, and retrieval of large files from legacy or deprecated systems, multiple performance optimizations have been implemented in the pro- posed search solution [6]. These optimizations enhance query re- sponse times, reduce computational overhead, improve storage efficiency, and maintain system scalability. The performance enhancements are categorized into data storage, indexing, query execution, and retrieval optimizations, ensuring that the system remains robust under high loads and large datasets.

Data Storage Optimizations

To reduce storage costs and improve retrieval speeds, files stored in HDFS are compressed using algorithms such as Snappy, Gzip, or LZ4 [7-9]. This ensures that large datasets occupy minimal disk space while remaining fast to decompress. Legacy systems often contain redundant data records, leading to excessive storage consumption. Dedupli- cation is applied at both the file level in HDFS to eliminate duplicate copies, ensuring optimal utilization of storage re- sources. HDFS is configured with optimized block sizes (e.g., 128MB or 256MB instead of the default 64MB) to reduce excessive metadata overhead while ensuring that files are efficiently distributed across nodes [1].

Indexing Optimizations

Efficient indexing plays a crucial role in maintaining high- speed query performance while reducing memory and CPU overhead. Custom language-specific analyzers, tokenizers, and stop-word filtering can be implemented to improve search accuracy and reduce irrelevant results. This is especially useful for processing natural language text in legacy documents. Bloom filters are enabled in HBase to quickly determine if a search index exists, preventing unnecessary disk reads and reducing query execution latency.

Query Execution Optimizations

Fast and low-latency query execution is essential for ensuring a seamless user experience. Frequently executed queries (such as searches for commonly accessed files) can be cached in Elastic Search, eliminating the need for redundant query execution. HBase uses block cache and region servers to store frequently accessed file paths, significantly reducing lookup time. Search results can be dynamically ranked based on keyword relevance, recency, and access frequency, ensuring the most relevant files appear at the top.

Data Retrieval Optimizations

Retrieving large files from HDFS efficiently is a critical aspect of the system, as traditional retrieval mechanisms can introduce significant delays. Asynchronous data fetching from HDFS can be implemented which allows users to start viewing or processing data without waiting for the full download instead of waiting for an entire file to be loaded. Large files can be retrieved using parallelized multi-threaded access, ensuring faster data loading speeds. All data retrieval operations can be logged and monitored, allowing administrators to track usage patterns and detect anomalies.

Conclusion

The proposed optimized search solution effectively ad- dresses the challenges of storing, indexing, and retrieving large files from legacy or deprecated systems by integrating HDFS, Elastic Search, and HBase into a distributed architecture. This approach ensures cost-efficient storage, high-speed indexing, and low-latency retrieval, enabling organizations to preserve historical data while maintaining scalability and accessibility. This paper presents several notable advancements that includes converting legacy system data into files and storing them in HDFS reduces operational costs while ensuring high availability and scalability, efficient indexing using Elastic Search by indexing key metadata fields in Elastic Search, allowing for rapid query execution, and integration of HBase as a lookup store enabling fast mapping of search queries to HDFS file locations [1-3]. This solution offers a practical and scalable alternative for organizations that need to store and retrieve historical or legacy data efficiently without maintaining outdated database systems. Despite its advantages, the system presents opportunities for further enhancement in advanced metadata extraction, AI-driven query optimization, enhanced security and access control and many others. As data volumes continue to grow, the demand for efficient, scalable search architectures will become increasingly critical. This research lays the foundation for future advancements in large- scale data indexing and retrieval, with potential enhancements in AI-powered search, real-time analytics, and cloud-based scalability [10-12].

References

1. D Borthakur (2010) "HDFS Architecture Guide," Apache Hadoop Project, [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs design.pdf.
2. S Gormley, Z Tong (2015) Elasticsearch: The Definitive Guide, 1st ed. Sebastopol, CA, USA: O'Reilly Media.
3. J Kunze, RP Lee, MD Riedel (2010) HBase: The Hadoop Database," Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, New York, NY, USA 1133-1136.
4. (2021) Amazon Web Services, Inc., "Amazon Simple Storage Service Documentation," Spring, [Online]. Available: https://docs.aws.amazon.com/s3/.
5. BSSK Chaitanya, DAK Reddy, BPSE Chandra, AB Krishna, RR K Menon (2019) "Full-text Search Using Database Index," 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA), Pune, India 1-5.
6. AK Mohideen, S Majumdar, M St-Hilaire, A El-Haraki (2020) "A Data Indexing Technique to Improve the Search Latency of AND Queries for Large Scale Textual Documents," 2020 IEEE/ACM International Confer- ence on Big Data Computing, Applications and Technologies (BDCAT), Leicester, UK 37-46.
7. M Blott, T Preusser, NJ Fraser, GK Kuzmanov, K Vissers

(2018) "A High-Bandwidth Snappy Decompressor in Reconfigurable Logic," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland 131-1315.

8. LP Deutsch (1996) "GZIP file format specification version 4.3," Re- quest for Comments (RFC) 1952, May 1996. [Online]. Available: https://dl.acm.org/doi/10.17487/RFC1952.
9. M Bart´ık, S Ubik, P Kubal´ık (2015) "LZ4 Compression Algorithm on FPGA," 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Cairo, Egypt 179-182.
10. Bennett E, Sako Dumnamene, Igiri Chima (2019) An Efficient Algorithm for Data Compression in a Distributed System.
11. PD Turney, P Pantel (2010) "From Frequency to Meaning: Vector Space Models of Semantics," Journal of Artificial Intelligence Research 37: 141-188.
12. SRK Saha (2017) "Natural Language Processing in the Digital Era," IEEE Transactions on Computational Social Systems 4: 81-89.