8-2021

MITRA: Robust Architecture for Distributed Metadata Indexing

Sarthak Thakkar
st4070@rit.edu

MITRA: Robust Architecture for Distributed Metadata Indexing

by

Sarthak Thakkar

THESIS

Presented to the Faculty of the Department of Computer Science

Golisano College of Computer and Information Sciences

Rochester Institute of Technology

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

in

Computer Science

Rochester Institute of Technology

August 2021

MITRA: Robust Architecture for Distributed Metadata Indexing

APPROVED BY

SUPERVISING COMMITTEE:

Dr. M. Mustafa Rafique, Chair

Dr. Michael Mior, Reader

Dr. Ifeoma Nwogu, Observer

Dr. Suhas Somanth, Dissemination Committee Member

Acknowledgments

I wish to thank the multitudes of people who helped me. Time would fail me to tell of ...

- to my Research Advisor **Dr. M. Mustafa Rafique**, for his well-oriented and enriching guidance throughout this research.
- to **Dr. Michael Mior**, Reader of the Defense Committee and Instructor of the Big Data course.
- to **Dr. Ifeoma Nwogu**, Observer of the Defense committee, for her promptitude and efficiency throughout this research.
- to **Dr. Suhas Somnath**, Dissemination Committee Member and a domain Scientist from Oak Ridge National Laboratory for constant guidance and support for this dissertation.
- to **Dr. Hans-Peter Bischof**, Program Director of M.Sc. in C.S., for his rigor and pedagogy. Instructor Advanced Object Oriented Programming class motivated me to apply concepts in parallel computing.
- to **Mrs. Cindy Wolfer**, Academic Advisor, for her assistance in diverse circumstances, especially in the final semesters.

– to **Dr. Sudharshan Vazhkudai and Dr. Hyogi Sim**, Scientists from Oak Ridge National Laboratory for giving directions for growth and research.

Abstract

MITRA: Robust Architecture for Distributed Metadata Indexing

Sarthak Thakkar
Rochester Institute of Technology, 2021

Supervisor: Dr. M. Mustafa Rafique

In the post-exascale era storage systems, a fundamental challenge faced by the research community is the efficient and scalable access to the stored information while meeting the high-performance requirements of big data applications. In this dissertation, we studied the limitations in the existing state-of-the-art architectures and proposed a system to address the challenges of scalability and high performance. Our proposed solution, called MITRA, supports several scientific formats, i.e., Hierarchical Data Format (HDF), network Common Data Form (netCDF), and Comma-Separated Values (CSV), and is composed of several software components that work together to provide high I/O throughput to user applications. The key novelty of MITRA lies in supporting a variety of file formats, generation and indexing of metadata for scientific datasets, and optimizing data lookup time while providing

scalability of storage subsystem with the increasing amount of data. MITRA generates and manages indices using a relational database which can be effectively accessed using conventional application programming interfaces (APIs). We evaluated the performance of MITRA and compare it with the traditional approaches for its ingestion speed, content processing, lookup time, and scalability for the generated indices. Our evaluation reveals that the rich metadata indices of MITRA improve system lookup by reducing the search space for the metadata that is not present in indices. Moreover, MITRA outperforms the existing approach in terms of scalability as indices grow in size by balancing the load between available hardware resources.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

With expanding volume of datasets and file archive storage, there has been significant progress in the development of file systems based on indexing files and effectively improving the time to discover the relevant data files. Realizing that current file systems can not cope with the diverse requirements of wide-area collaborations, researchers have developed data catalogs to meet their needs [43]. However, the key challenge is the explosion in the volume, velocity, and variety of data for research communities composed of multiple disciplines. This makes it challenging for anyone, especially in the highly heterogeneous field of science, to find and discover information easily. Information is discoverable only when key attributes about it are readily available. In the case of scientific research, this means capturing meta information such as experiment parameters, the intent of experiments, presence of or key characteristics of the raw data, authors, related papers, etc. In such cases metadata provides a means of indexing, accessing, preserving, and discovering digital resources [5]. Metadata is an invisible infrastructure for information attached to files [30]. Nonetheless, extracting and maintaining an index of highly heterogeneous metadata which varies from community to community even within

the same discipline has turned out to be an arduous and intricate task if we are talking about petabyte-scale systems.

At present, scientists tend to cram any and all metadata that they deem important into the file name since the file systems are incapable of capturing such rich metadata. An ideal distributed file system would provide all its users with coherent shared access to the same set of files and high performance to growing user community [41]. However, scientific dataset users do not only read and write data, they also manipulate metadata to organize their data in a typical tree hierarchy [16]. Because file system metadata operations make up as much as half of the typical file system workloads [34], effective metadata management is critical to overall system performance [42]. Dedicated databases seem like a better solution instead of relying on the limited capabilities of file systems alone.

The motivation of this thesis is to design and quantify a system that streamlines the process the file ingestion to high volume archival storage and to process as well as generate metadata from files to make a rich metadata index. Accessing the files in archival storage using the metadata index generated can improve the lookup time for such high volume storage systems significantly. This system is designed to meet the increasing demand for collaboration among the research communities and collaborations for multi-disciplinary research.

1.2 Problem Description

In the scientific community, the archival storage [11] is composed of petabytes of data. Files being on archival storage most of the time the only way to access them is by Network File System(NFS) [29] where an NFS server exports a file system hierarchy to the NFS client for mapping it to local namespace. The growing size of modern storage systems is expected to exceed billions of objects, making metadata scalability critical to overall performance. Many existing distributed file systems only focus on providing highly parallel fast access to file data and lack a scalable metadata service [32]. Accessing the data from those archival storage based on a certain attribute for related research is an arduous job. To search for specific datasets based on values of certain variables requires the system to go through the entire archival storage in order to return the requested datasets. Consider an archival storage with a few million files of atmospheric data and for a related research if a scientific data user wants to access the files which contain temperature in a certain range or the data for a certain range of coordinates for a certain time interval. The user will have to scan all the files in the given archival storage and go through the entire content to fetch the required files. This effort to crawl through high-volume data is challenging.

To, further illustrate the problem scientific datasets many times are in different file formats which are used by the scientific community to exchange the data. The present-day high-volume file management systems provide limited support to efficiently store the files and later access them using the meta-

data for the scientific file formats. Also, all the systems we studied act on already present metadata for a given set of files and not on processing them to generate metadata for a richer index.

1.3 Research Objective and Contribution

In this thesis, we are focusing on increasing the accessibility of files in archival storage using a metadata-rich index. One important point to note is archival storage contain high volumes of data accessed when needed. So, the file system generally used for archival storage is POSIX [21] file system. The architecture proposed in this thesis is an additional tool that can be used with a traditional POSIX file system and enable easy access for the containing datasets. Below, we present the design objectives of this work.

1.3.1 Metadata Generation

Besides with indexing the available metadata attached with the file, this architecture also allows us to generate metadata based on selective processing of the data. Apart from previously attached data searching for a set of files can be further improved by added more features about the given dataset. This system is designed by focusing on scientific file formats. Scientific file formats are different structures of datasets structured as either multidimensional rows and columns or hierarchical data but the data would be numerical. So while ingestion of a certain structure of dataset with specific parameters for certain discipline we can also add them as metadata. For instance, if a

scientific dataset user from the climatic research division wants all files containing temperatures in certain temperature ranges or files for certain ranges of geographic coordinates. If we have indexed files while ingestion It would make it much more efficient to access the respective files.

1.3.2 Scalable Architecture

For processing high volumes of storage, this architecture is designed to be scalable or a given cluster of nodes. It is configurable over a heterogeneous cluster as well allowing it to distribute the load based on the capacity of each node. Scalability enables the handling of high-volume data.

1.3.3 Custom File Format Interface

For adding a custom scientific file format an interface is defined in the proposed system. So, just by implementing the interface one can easily and fill in the basic functions to open and read a file. This makes it easier for scientific data users and scientists to easily add a custom file format metadata reader and extractor to an existing solution.

1.3.4 Flexible database schema management

The proposed system identifies the file extensions in the given file system and uses a specialized extractor for the specific file extension and updates the metadata tags dynamically. While processing files of the same extension, if it discovers new attributes for that file extension, then it alters the database

table for that extension and adds new fields automatically in the database table. Furthermore, it generates tables for the supported extensions if they do not already exist in the database. Allowing it to update the database while processing the files and managing concurrent database updates. the proposed system also identifies and eliminates redundant files from the same directory location and does not index the same file over and over again.

1.3.5 Multilevel Key-Value Pair Metadata

The proposed system also stores and indexes a file if the previously attached metadata contains multilayered key-value pair attributes. Client nodes map them to tables in Relational Database Management by using the parent's names as pointing references and a level identifier for every level of keys followed by the actual metadata key with the row containing file reference and given cell containing the metadata value for the metadata key.

1.3.6 Batch Processing

To maintain the optimal performance of the client node and override the possibility of network buffer overflow while processing a high volume of datasets the proposed system uses batch processing. In batch processing, each client node has a defined batch size based on the hardware configuration of the node. So, when the number of files assigned to a client node is higher than the batch size then the files sent to the client will be in the given batch size specified by each client node.

1.4 Some Important Terms

During the manuscript we are going to use terms stated below while talking about the working and design of the system. These descriptions below provide the context in which those terms will be used.

1.4.1 Generated Schema

A generated schema is a set of key-value pairs that contain the unique structure of each type of file. For simplicity, it can be understood as if in a CSV file format a file is added every day containing atmospheric pressure of an area and another file added every day which contains the atmospheric temperature information. For both these files, the file format is the same but the columns (i.e. structure) of the files are completely different. Also, The columns of each of the types of this file would be the same with just different values of rows. So, based on the structure of the file we can generate a schema of the structure, and all the files matching the schema can be labeled and clustered with the respective schema_id. In the above example, we had CSV files so only columns were used as structures. But, For files like netCDF, we have a combination of dimensions and variables in those dimensions, For HDF files we have hierarchical order and attributes of each key in hierarchical order and for a custom file format it can be defined as well.

1.4.2 Derived Schema

A derived schema contains information about the new metadata that has to be generated and added to the metadata index. A derived schema should be specified in form of a JSON file specifying the operation to be performed and file name to get the generated gen_schema_id of that file and apply to all the files of that signature and update them in the database.

1.5 Layout of thesis

The remainder of the manuscript is organized into 5 chapters. In Chapter 2, we discuss about the contemporary systems and proposed frameworks in the direction of organizing files using metadata. Chapter 3 describes the architecture of the system to be mounted on a POSIX file system and database design. Chapter 4 presents the detailed description of every step of the control flow in the system to the smallest detail possible. This section provides a core understanding of our proposed system and it's working. chapter 5 shows the settings of conducted experiments and the outcome of the experiments, analyzes and discusses the results and learning from each experiment. Chapter 6 contains a detailed description of optimizations done through the journey of developing the architecture and the impacts of those optimizations. Chapter 7 concludes the thesis with the note of the future work and conclusion of the thesis study.

Chapter 2

Literature Review

In order to improve the lookup time for high volume storage systems, there has been extensive research going on in the scientific community. Multiple approaches have been proposed to simplify the indexing and processing of scientific data format files to manage large archival storage. However, these approaches have their limitations in addressing discovery and retrieval for scientific files. No concrete system has been found to address all three aspects of the problem: flexibility of file-format processing and schema-based pre-processing, and a well-balanced distributed frame for scalable high-performance computing. In the sections below we have discussed similar approaches and their limitations.

2.1 Hadoop File System

Hadoop File system (HDFS) [36] is widely used for handling high volumes of data. It helps in the distributed storage and processing of files effectively. However, Hadoop can process only certain scientific data file format which are follow key-value store such as SequenceFile [26], NLine [13], KeyValue, FixedLength, etc. [7]. Due to which it only allows file types like HDF5

or CSV [27] to be processed and accessed. For, given file types it supports Map-Reduce[15] jobs to parse and process data.

Due to the limitation of processing only key-value paired data, it leaves out the processing for file formats such as netCDF [7] over a distributed frame. To solve the problem of processing netCDF files over Hadoop file system solutions have been implemented of a netCDF file to CSV files and then ingest in Hadoop file system for processing [7] or other solution to use netCDF files in Hadoop file system is to use SciHadoop [10, 12]. With workarounds possible to use multidimensional array file formats in Hadoop. Hadoop still lacks to address explicit processing of data to generate more metadata from files and index it along with default indexing of metadata in name node.

2.2 MetaStore

MetaStore [31] is a NoSQL-based framework to manage scientific datasets using the attached metadata of the files. This is a flexible framework designed to enable indexing, storage, and retrieval of scientific files. This framework uses a NoSQL [38] database considering every file to have a unique set of headers for metadata and might require unique keys for each file record. Every unique set of metadata headers is addressed as a new schema for this architecture. It uses the Server-Client model over a distributed network to process the set of given files where the server is used as a load balancer to manage the distribution among the clients. Being a NoSQL based architecture it supports key-value paired hierarchical file formats such as .hdf5, .kdf [3], and .tiff [2].

In order to work with files of multiple schemas, it requires a pre-registered schema definition in form of JSON/XML the files matching the schemas from the schema registry are processed and stored in the database. It also supports multiple databases such as ArrangoDB [1] for storing both key-value pairs and graphs or it can be used with a combination of MongoDB[28] and Neo4j [22] for storing metadata. Later allowing a full-text search retrieval REST-API for better accessibility.

However, the shortcomings of this system are It requires manual submission of each schema file and attribute to be processed, it lacks the support of multi-dimensional arrays file formats such as netCDF or tabular files like CSV. It also lacks the ability to generate more metadata by processing the interesting attributes from the contents of files and store it as a part of metadata.

2.3 MetaCat

MetaCat[6] is a similar approach for metadata-based management of files using RDBMS as database storage. It also proposes a system based on the schema of the files to be processed. Metacat is a proposed system to manage files with multiple storage structures on a heterogeneous scalable database cluster. It first creates an XML file of metadata of the ingested file and inserts it in the database. It also suggests a custom query language to retrieve the files using stored metadata and allowing to query over hierarchically stored XML data in RDBMS. It provides user-based access control and authentication and

reliable replicated storage to avoid data loss.

This model does not support libraries for reading previously attached metadata or perform content processing for metadata generation. It extracts metadata and generates layers in a hierarchical model based on the access and storage information of files using trivial information such as file path, file name, and time information of file properties. Due to these features, it becomes useful for all types of data files in the file system. However, the trade-off is limited exploration of every file type makes it easier to access later.

2.4 SoMeta

SoMeta [39] is a scalable Object-centric metadata management solution. This approach is based on storing and retrieving metadata information of files on an Object-oriented database. It proposes a scalable architecture that can effectively handle rapidly increasing files and metadata using a high-performance cluster of computing nodes. SoMeta proposes a decentralized tagging approach for storing metadata. This solution is based on utilization of OpenStack Swift database storage[4] and DAOS[25] as processing platform for the establishing object-centric metadata library. To improve the performance SoMeta uses flat namespaces for effective management of Distributed Hash Tables (DHT)[14] and bloom filters[40] for maintaining unique Metadata objects across all the servers. Object-centric storage also allows the solution to be flexible as attributes in form of key-value pairs can be inserted, updated, and deleted as required and retrieve efficiently with bloom filters over DHT.

As per the proposed system of SoMeta, it is focused on effective metadata storage and retrieval of metadata on object-centric storage compared to existing RDBMS and NoSQL databases. The shortcomings of this approach is it designed for object-based storage systems like RADOS, Amazon S3, and Openstack swift. MITRA is to support archival storage which is mostly based on traditional POSIX file systems. Also, as it involves only the storage and management of metadata. There is no Address of support of various file formats for metadata extraction. It also lacks the component of processing the ingested file contents for generating more metadata and improving the accessibility of the file with a richer metadata index.

2.5 TagIt

TagIt [37] is a user-defined metadata-based tagging approach for a distributed file system. TagIt is one of only a few solutions that we encountered which was designed on the notion of processing the contents of the data files and generate more metadata and add it as a tag along with metadata to improve the accessibility of the file later on when required from an exascale storage archive based on certain features of the file. This system is implemented on GlusterFS[8] as a shared-nothing distributed file system as well as CephFS[42] for an object-based storage file system. It allows simultaneous access to the database as the database is horizontally shared among all the worker nodes by using sharding. It also focuses on fault tolerance and durability by checkpointing the shared database to persistent storage to recover

quickly in situations of database failure. It achieves improved performance for search queries over distributed architecture by indexing the tags in a shared database.

This approach is primarily focused on generating tags and systems to manage the tags over Relational database (RDBMS) and Object Storage Daemon (OSD). In the proposed system the analysis of the system is provided for a directory containing netCDF files. However, the description of components that can be used for the generation of metadata supporting versatile tag generation from data files of file formats is yet to be addressed.

2.6 Empress

Empress [24] was initially designed as a robust system to support scientific file formats and index files using metadata over a distributed cluster. Later on, in the releases after that, it evolved to support extraction of user-defined metadata based on the contents of the file to improve accessibility. Empress is designed for users to perform multivariate analysis, atomic operations, fault tolerance, and scalability. Empress is based on In-memory RDBMS as database storage and has shown promising results compared to the Hadoop file system and SciDB[9]. It reads data directly from simulation and processes it for basic metadata and custom metadata specified by the user.

Empress is a similar approach to MITRA in terms of a scalable and distributed framework for the generation of metadata and index it. The limitation of this approach is it ingests the data directly from simulations and

processes it for generation and indexing of metadata and then forward data to be stored in the file format it was supposed to go. This approach can generate a rich metadata index but it is applicable only for the data generated after implementing this system for future experiments. MITRA allows the generation of a rich metadata index from data stored in archival storage. Apart from that MITRA also allows scientific data users to add support for custom scientific file formats.

Chapter 3

MITRA Design

In this chapter, we'll be discussing about the component design for MITRA from a system-level perspective, database perspective, and design of a tree-based load balancer implemented for MITRA.

3.1 High-level System Architecture

MITRA is a distributed framework designed to run in a physical, virtualized, or containerized environment that can span multiple nodes. The formation of MITRA consists of a distribution server, a database server, an API server, and multiple client nodes. Here, we have 3 separate servers for Database, distribution, and API requests. However, it is not necessary for all the servers to be on the same physical node. The target file system or directory that is required to be indexed is mounted on each client and the distribution server functioning as a master node. The distribution server will then analyze the target file system and distribute its contents in chunks based on the available client nodes.

Each client node processes the chunks assigned by the master node. Initially, a client processes the previously attached metadata to the files. Then,

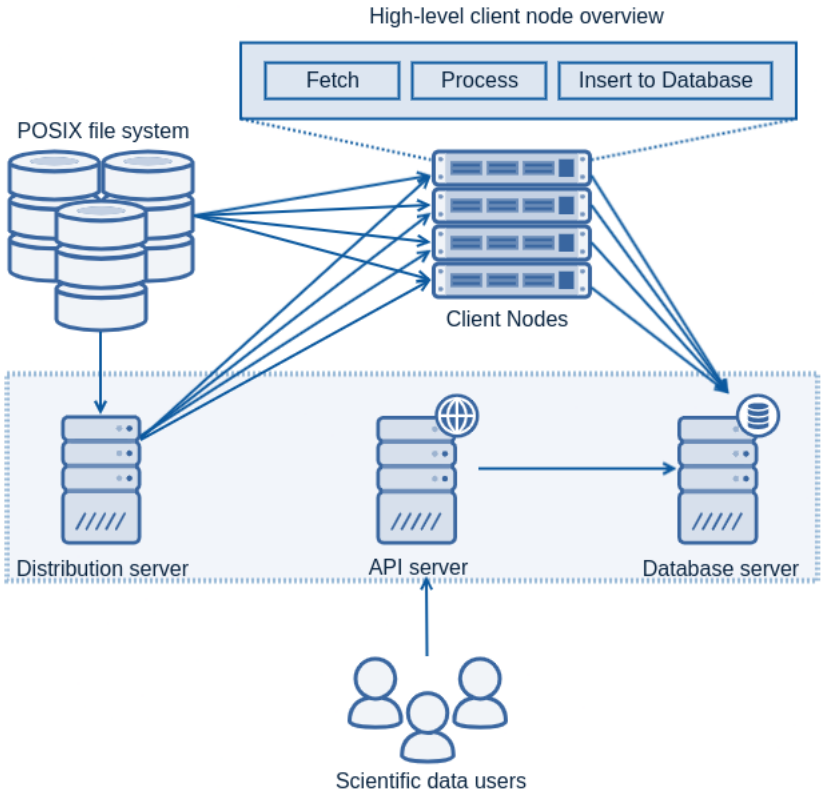


Figure 3.1: High-level system architecture

a client node will further process the contents of the assigned files. Processing the contents to generate metadata can be performed at a later point in time as well. Generating metadata adds metadata based on the content which can be specific to the schema of the given file, e.g., a date range, a temperature range, and data related to some geographical area. By indexing the metadata in SQL server it enables easier and efficient file lookup based on a specific attribute.

3.2 Database Design

MITRA uses a RDBMS database PostgreSQL for storing the metadata and index it. This database was specifically chosen as it allows storing fields in JSON format as key-value pairs. Also, its transaction management and concurrency management works well with multiple simultaneous connections. The database consists of a basic map index table that is the common table with a shared attribute to directly access metadata and access information. Apart from the common file_info table, there is a set of three other tables for each file format type which we want to process using the architecture. This set of three tables can be described as:

1. **Metadata Table (`meta_{file_format}_info`)**

   This table contains all the metadata attributes of files of the specific file format. Every column in the row will be a metadata attribute. A column will be added for every new metadata attribute discovered for the file format `meta_{file_format}_info` table is associated to. Every row contains values associated with metadata attribute associated with the file. Each row has a unique identification field `file_id` which will be directly mapped to the `file_info` table.

2. **Generated schema Table (`gen_{file_format}_schema`)**

   This table contains the generated schemas for that file format. Along with `gen_schema_id` for each schema and the columns for the structure of that file format. Also, there is a column with an array of values to

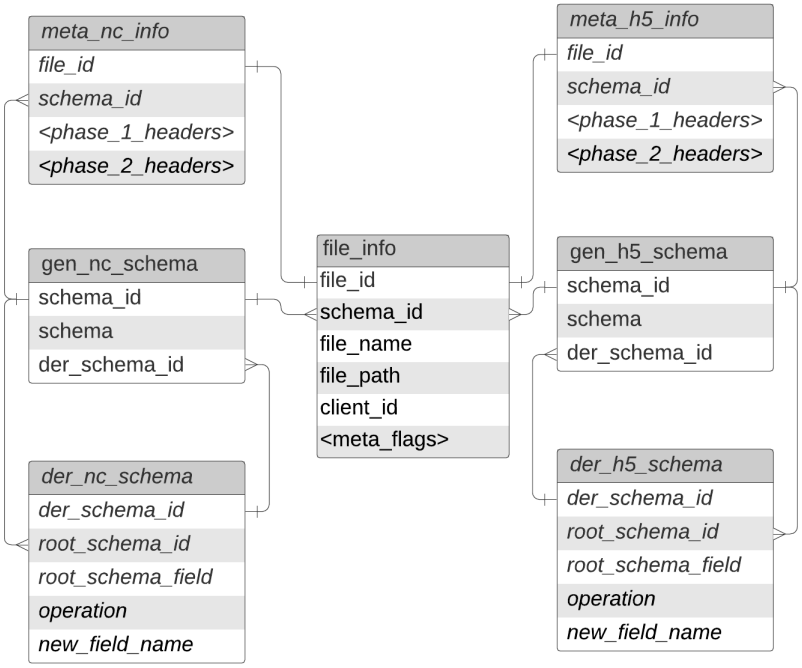


Figure 3.2: Database design for MITRA

be derived schema id's (i.e. `der_schema_id`) that are mapped to that generated schema.

3. **Extraction schema Table (`der_{file_format}_schema`)**

   This table contains information about the fields to be processed and added to the metadata table. The id of the generated schema mapped from filename and also the new name of the field to be named after performing a certain operation during ingestion of file.

3.3 Heuristic Tree Distribution

Heuristic tree distribution is used for the balanced distribution of given files across the distributed network. MITRA does load balancing using its own data structure. Here, the master node analyzes the file system and assigns a heuristic weight to each file and directory by generating a tree data structure. The heuristic weights can be calculated based on various factors, such as file size, extensions, and the number of files under a specific directory. The resulting tree is further divided into sub-trees for distribution to the available computation nodes. Each sub-tree contains the file paths of the mounted directory. Based on the number of client nodes connecting to the server the distribution server node will allocate a sub-tree to each client node such that all clients get sub-trees of either the same or similar heuristic value. After sub-trees are allocated, they are sent in small batches to the client nodes for processing based on the processing power of each client, which is specified in the configuration file.

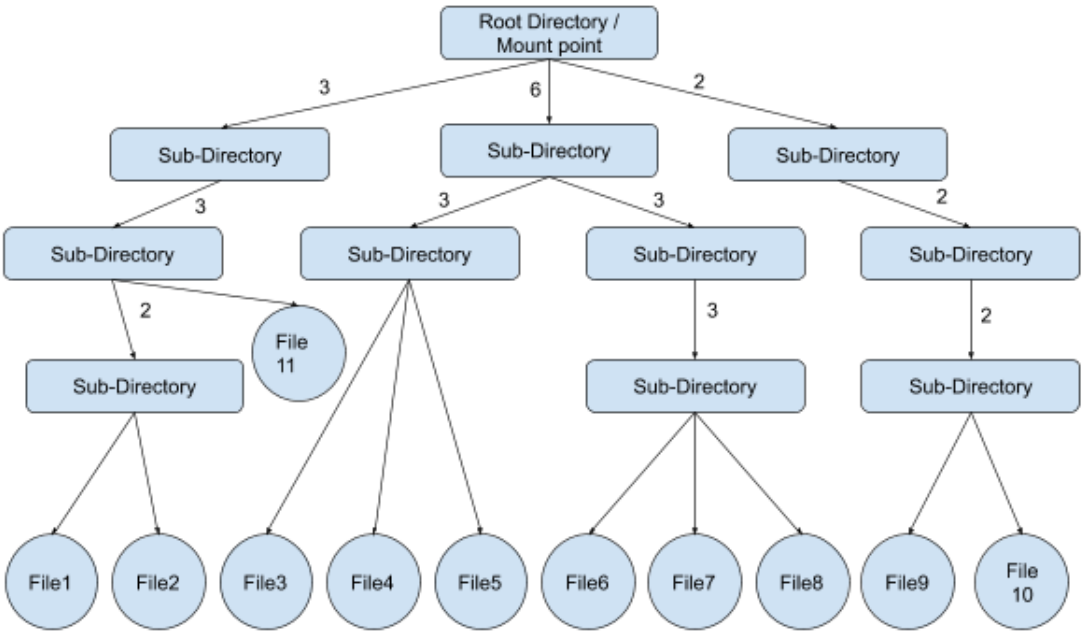


Figure 3.3: Heuristic tree structure for example files

Chapter 4

A Method for Metadata Generation and Indexing

In this chapter, we'll be discussing the flow of control through MITRA and understand the processes happening while discovering the files on storage, accessing the headers, generating an index in the database, and processing the contents of files for metadata generation.

4.1 Metadata Harvesting Process

System Flow can be understood starting from fetching a file from a POSIX archival storage to a two-phase ingestion and database management. The ingestion of files in the proposed system and the processing of the file is the core using all the components we discussed in the above sections. Once the system starts processing the files. Files are processed in two phases. In the first phase, the system creates the basic layout and stores the previously attached metadata and marking files by generated schema labels and segregating them based on the features given. In the second phase, the system processes the dereived_schemas provided related to the files to be processed and respective operations. In this phase system then processes the contents of

the selected files and generates metadata and adds the generated metadata to the previously created metadata tables.

To design a balanced, scalable and distributed framework using client-server model we distributed the server and distributed client cluster to get optimal and balanced performance with high volumes of data. Here we discuss the overall flow of system and functioning.

In the beginning of the system, the server prepares a heuristic file tree for the given storage for load balancing when clients are connected. the process of creating a tree is quick and efficient and is discussed in the evaluation section. After preparing the tree. It waits for clients to connect once the clients are connected. It prepares the chunks of files to be processed by each client and communicates the file distribution in batches to the client nodes. Once the client receives the list of files. It starts a multiprocess activity to prepare metadata, insert it into the database, and map schema. After all client nodes complete phase 1 processing. The server scans the derived schemas present in the storage. It then inserts the schemas into the database and passes on the list worker nodes to map the derived schemas with the root schemas. Mapping is a computation demanding process so it is distributed over client node cluster. Once Mapping is complete server node access the list from the database and distributes the list evenly among the client nodes for balanced networks and computational load for metadata generation. Phase 1 and Phase 2 are described in detail in the below sections.

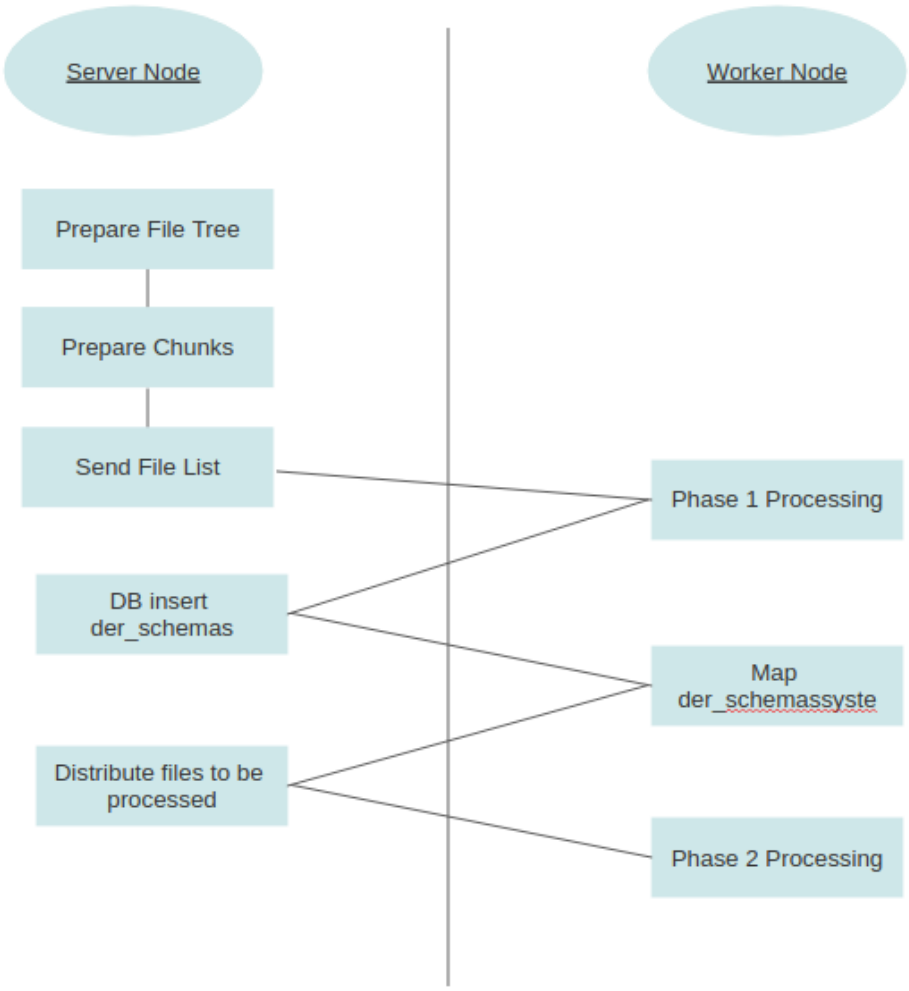


Figure 4.1: System flow design

4.2 Phase 1

In this phase the file processing will start by the server analyzing the file system assigned to it and prepare the subtrees containing file and folder paths to be processed for each client. These subtrees will then be sent over to clients respectively in small sets at the rate that the client processes the assigned files. From the client's end, As soon as it starts receiving the chunks of subtree assigned to it. the client starts by accessing the file format of the assigned file and select the appropriate extractor. Then the file is accessed by the extractor of the format and reads previously attached metadata leaving the contents of the file untouched. As it completes reading the previously attached metadata for a given chunk it sends the object to the database manager. The database manager then reads the metadata received and verifies if it already exists in the database. If it is new data the database moves forward with the database insertion phase and generates the dynamic queries to insert the metadata in tables. Our Database management is flexible being RDBMS. So if a new attribute is discovered for a file format a new column will be added and then the insertion operation would be processed.

After successfully inserting the collected metadata of chunk into the database. The client then requests the next chunk from the server for completing the ingestion of all the files in the subtree assigned to it.

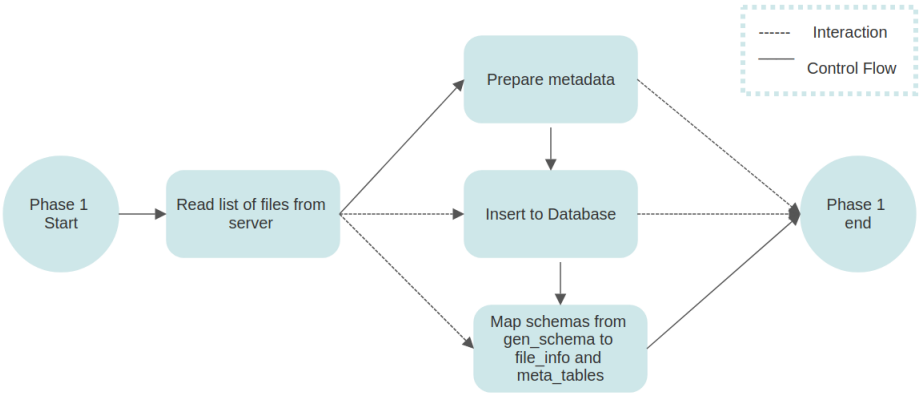


Figure 4.2: Phase 1 processing

4.3 Phase 2

After the previously attached metadata is read. This step focuses on the generation of new data for the schema of files that was requested while ingestion. After all the clients complete phase-1. It looks for JSON files specifying the interesting data specified to be attached as metadata. Once it discovers the JSON files for the schema to be worked on an operation to be performed on.

The server distributes the JSON files across the clients. The JSON file will contain either the generated schema id if the two phases are done in parts or a filename of the generated schema id type. If the JSON file contains the filename then the client will find the generated schema id from the file_info table created in the first phase and Insert the values of JSON file in the der_file_format_schema table as well as update the der_schema_id in gen_file_format_schema table. This will complete the mapping of all the

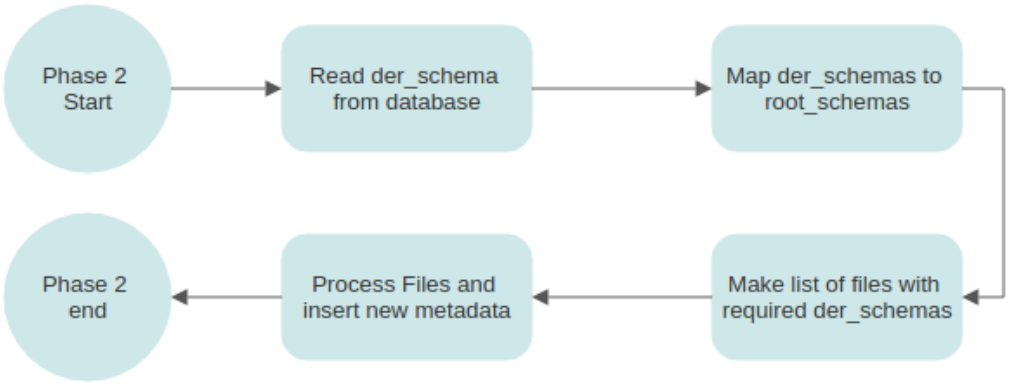


Figure 4.3: Phase 2 processing

user-defined schema for custom metadata extraction to be mapped to all the necessary file indexes.

After the mapping is complete. All clients will again report to the server and wait for further commands. The server will then form a list of files to be processed and the number of operations to be performed on each file to generate the new metadata fields. This list of files is then shared over to clients in the form of the file_id and derivation schema information to the clients. At this stage clients access the files based on the file paths indexed and database and fetch the contents of the file to memory using the extractor defined previously for that file format. Once it loads the part of the file it wants in the memory. It performs the requested operation to generate new metadata. Once the metadata is generated the client will then access the metadata table for the file format of the file and add a new column for the

specified metadata and add it to the index.

As all the clients complete updating the metadata for the files and operations assigned to them the file ingestion based on metadata indexing is completed. So, the next time when the list of files are queried the newly generated metadata can be helpful for doing an efficient search of files based on the summary of the contents of the file.

4.4 API access

To store the data in a form so that any scientific data user interested in retrieving the data, later on, can easily do it. For storage, we used a Relational Database Management System (RDMS) PostgreSQL. Whereas for the retrieval system, we used a Flask-based REST API[19] server for querying the database as described in the system overview. Using the APIs user can request files of certain geographic locations, or files containing a value of a variable in a certain range as well as can send a request to download the files matching the criteria to the devices sending the request. In this thesis, we have focused more on efficient ingestion and indexing of the given data to ease the access for retrieving it later. API server we used here is a single process-based server used for simple range queries and get files. It can be improved later on the create a multi-processed scalable API server as well to access data from the database more efficiently and can also be integrated with cache-based storage and access in future improvements.

Chapter 5

Performance Evaluation

Multiple experiments were performed to understand MITRA's performance from different perspectives. To evaluate the performance of MITRA, we tested the system against varying numbers of files and the number of distinct schemas to measure impact on the database, time taken to develop a heuristic file tree over an NFS file system, and to observe the behavior of scaling when different loads ingested.

To benchmark the overall performance we tested it against a very straightforward and naive approach and limiting MITRA to a single file format. The naive approach is a simple approach designed to act in a similar way but in a very simplistic and straightforward aspect with minimal overheads. Naive approach processes files in phase 1 and phase 2 as well following the same steps to fetch files in phase 1, read the header, and update the database. Similarly, In phase 2, the system fetches files, processes them, and updates the table. While designing naive approach the differences we considered were naive approach was a stand-alone script not following the client-server model to process the given storage. Also, the database was simple where there was only one table to manage metadata of given files. As the idea of the naive

approach was to keep the complexity of the approach to a minimum to get the best performance the functions to access metadata and contents of a file were limited to one file format. Considering the naive approach for different file formats we would need different libraries to access data for each file format.

For this evaluation, we have used NetCDF4 (.nc, .nc4) as the only file format of files to be processed. To understand the performance of MITRA against a naive approach we tried to profile the overall time taken by both approaches in the multiple steps that are performed while ingestion. The detailed analysis can be understood in the below sections.

5.1 Experimentation Configuration

For experimentation, we utilized a cluster of 7 computational nodes each node containing 12 virtual cores of Intel Xeon 3204 processors, 32GB of main memory, and 1TB of storage. Out of the 7 nodes we used one node as a distribution server and 3 nodes to be client nodes hosting the clients. The remaining 3 nodes were used as NFS storage mounted on the distribution server as well as client nodes. On client nodes to experiment with a different number of clients, we launched a docker for each client node in the experiment restricted to use a certain amount of CPU and main memory. For Relational Database we used PostgreSQL database set up on of the server nodes.

5.2 Scalability Analysis

In order to observe the performance of the system over distributed network cluster as well as increased loads for a static configuration, we performed scalability tests in two parts. For the first part, we used the fixed load of files and operations and conducted multiple iterations of experiments by increasing the number of clients in each iteration and observed the time taken by the system. We did this set of experimentations for both Naive and MITRA. For the second part of the experiment, we performed the experiment by increasing the file loads to be processed and plotted it across the increasing number of clients to observe the effect of scaling file load on MITRA.

5.2.1 Scalability Against Naive Approach

To see the efficiency of resources used by both approaches we scaled the computation resources over the specified cluster. for this experiment, we compared the time taken with the Naive approach as well. As the Naive approach is not scalable we manually divided the files into a number of divisions as the number of client nodes. By this, we ensured that the files processed by each client in MITRA, as well as the naive approach, remain alike for a fair comparison. In the earlier part of development when we measured the performance of MITRA over the naive approach on a single node there was a significant difference in the time taken for a single node. As MITRA creates a lot of overhead when it comes to database management maintaining tables and communication between server and client for managing concurrent

access to database and file servers for parallel execution. On the other side, the naive approach performs with a minimal set of operations and complexity. But as we scale the number of nodes assigned we can see in figure 5.1 that the extra efforts put in by MITRA are actually assisting in managing the network traffic load, concurrent database management, and even utilization of given resources. Which results in a great improvement in performance until the physical limits of resources and reached. Once the Minimum time taken for a client node to respond after processing the files assigned depending on network bandwidth available or computation time and resources to process the files after fetching and inserting them into the database is reached performance becomes constant. If a performance bottleneck is reached in scaling it can be solved by improving the network bandwidth shared or computation response time of client nodes or changing database concurrent access policies. With the present experiment configuration as shown in graph 5.1, we observed at 12 clients the improvement in performance became minimal and we reached the constant performance at 18 client nodes for MITRA. After 18 nodes as we increased the nodes no change was observed in terms of performance as we reached the network traffic limit. We can observe that in the next experiment on section 5.2.2 where we increase the number of files and time taken to process increases on the y-axis but the trend with respect to the x-axis remains constant.

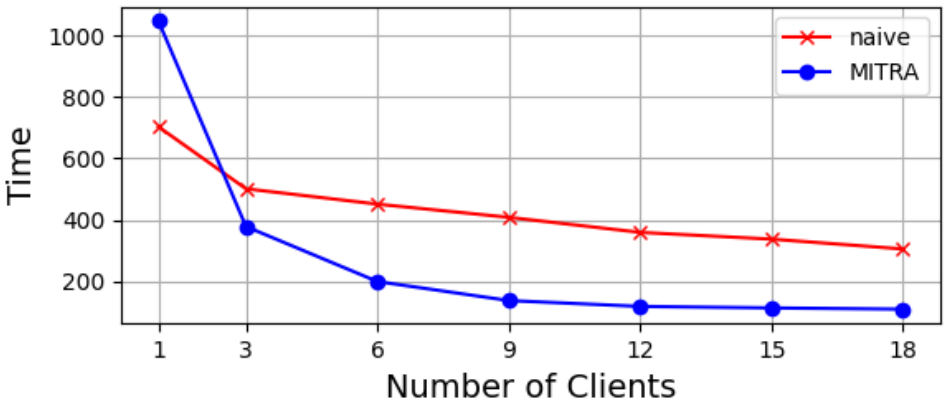
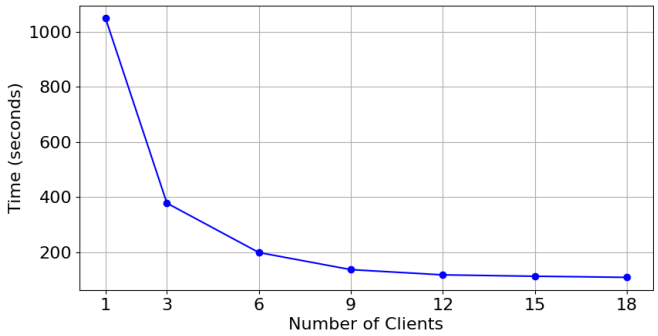


Figure 5.1: Scalability performance of MITRA against Naive approach

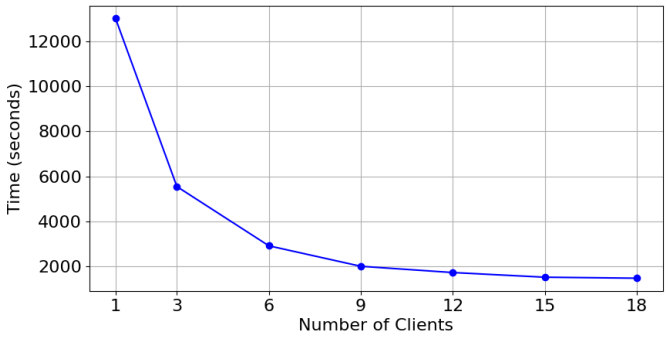
5.2.2 Scalability Analysis for Increasing File Quantity

To understand the behavior of utilization of physical resources from the previous experiment. We conducted the experiment with increasing file quantities. When tested against 3 different collections of files with a different number of files in each collection. The performance curve in figures 5.2 (a), (b), and (c) remained similar showing an increase in time taken with increasing in file load but the improvement in performance was the same as the previous experiment. Where a significant dip was observed as clients increased to 3 and

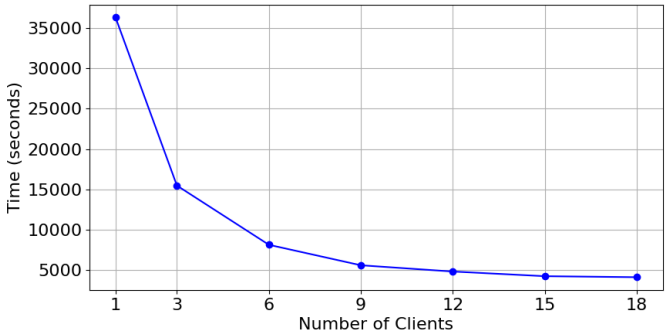
became almost constant at 12 clients. From this, we can understand that the bottleneck observed here is network file transfer bandwidth. If computation overhead was present then with an increase in the number of files the constant performance would shift towards the right for each graph where the almost constant performance would be reached with higher computation nodes. So, we can say that the bottleneck here is network file transfer bandwidth. This means the scaling was aligned as the performance observed was executing a similar behavior just scaled to higher numbers based on the increase in the volume of datasets to be processed. If there had been a problem with memory mismanagement or concurrent fetching and processing of files the performance graphs would have been steeper with increasing file loads. So, We can say that increased file loads would cause an increase in time taken for execution but the best performance in resource utilization is reached for the same number of client nodes per server in a given configuration although file quantity increases significantly.



(a) For 20K Files



(b) For 100K files



(c) For 200K Files

Figure 5.2: Scalability performance across various loads

5.3 Schema Variation Analysis

To analyze the performance overhead caused by different types of schemas of files for the same number of total files we tested our system with five different sets of data with different numbers of distinct schemas. For this experiment we have results from MITRA only as the naive approach can not differentiate and label files based on the schema of the file. As we discussed in the above sections that every time a new metadata header is found the meta_table of that specific file format needs to be altered for all the present rows in order to accommodate the new value in the given table structure. So, In this experiment, we analyzed the performance impact of altering the tables in databases by increasing the number of distinct schemas up to 100 times and observe an increase in the database operations to be performed.

For Phase 1, As observed in figure 5.3 the difference in performance was expected to be scaled up as if the increase in alter table query is increased up to 100 times the time taken to perform those operations and process the given data should be much higher but as it can be seen in given graph that even if the increase in the number of schemas is 100 times higher the time taken to accommodate those schemas for phase 1 shows less then 2 times the increase in time taken. By this, we can say that the cost of altering metadata table and mapping schemas for different types of schemas increases quite slowly compared to the increase in the number of distinct schemas. It can be said that for different types of schemas for a given set of dataset it doesn't impact substantially on performance for ingestion of the datasets.

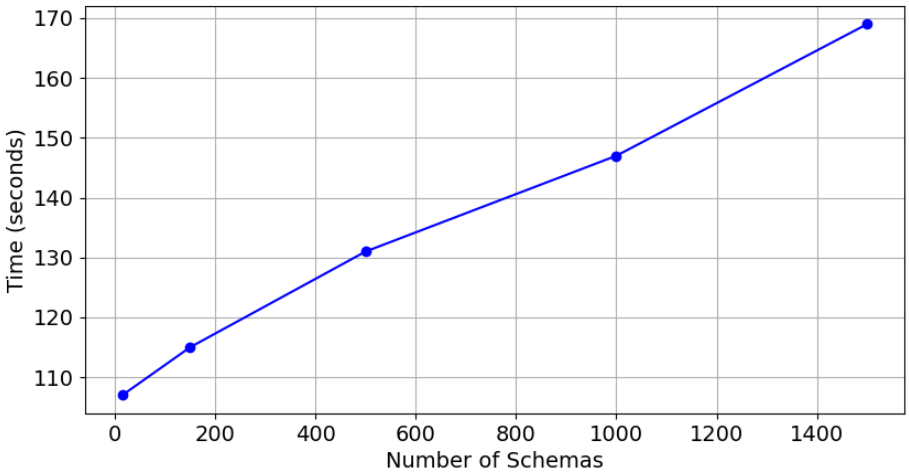


Figure 5.3: Phase 1 performance against various schema types

For phase 2, As observed in figure 5.4 the task in focus is to fetch the contents of relevant files and perform the given operation (calculating mean) in order to generate metadata and store it into database index to access datasets through it at a later point in time. The most substantial task in this phase is performing the update query for every relevant file processed and attaching generated metadata in the index. In this phase results, we observe that even if the distinct number of schemas are scaled up to 100 times to fetch, process, and update the impact on the performance of the System is approximately 2 times. By this, we can say that the cost of fetching content, processing, and managing database for different types of schemas increases quite slowly compared to the increase in the number of distinct schemas. Also, for phase two it can be said that the impact of types of schemas is considerably less on

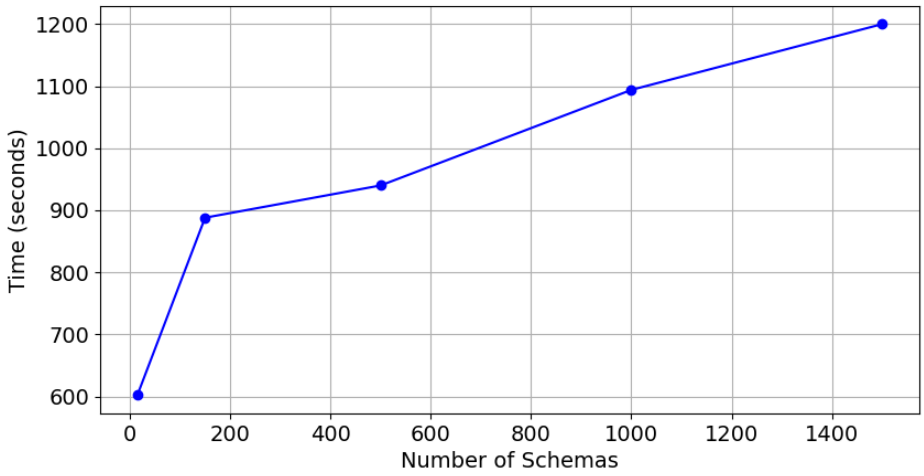


Figure 5.4: Phase 2 performance against various schema types

overall performance.

5.4 File Quantity Load Analysis

To observe the performance pattern of the system under different file quantities we used different collections of files. Where each collection of files had a different number of datasets with the constant number of schemas as other collections and plotted it on graphs 5.5 and 5.6. The primary reason for this experiment was to see if the system is causing any extra overheads with increasing file quantities. The impact of increasing file quantities could increase context switching and overloading the main memory utilization or buffer overflow between worker and Database server. If there would have been any of the given conditions the trend of performance against the time taken measure would curve up towards we tested it from 20K files to 500K files but

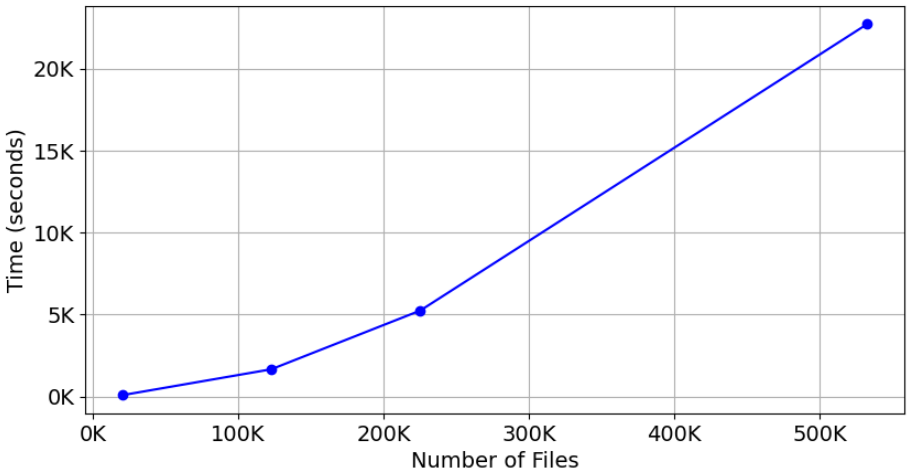


Figure 5.5: Phase 1 performance against various file quantities

if we observe this performance in the graph it seems like it is more close to the linear performance. From this, we can tell that the system performance against various file loads scales linearly without causing any extra overheads due to an increase in the number of files.

For phase 2, The main-memory overload and traffic overflow were primary overheads were kept in mind with an objective to see if increasing file loads causes any of the overheads and increases the time taken for increased files significantly. As in this worker nodes loads the content from the files in memory and then perform the operation on a data present in memory. So, we used files up to a file size of 1 Gigabyte but as we observed the time taken was scaling linearly without causing any extra overhead and creating a significant change in the trend.

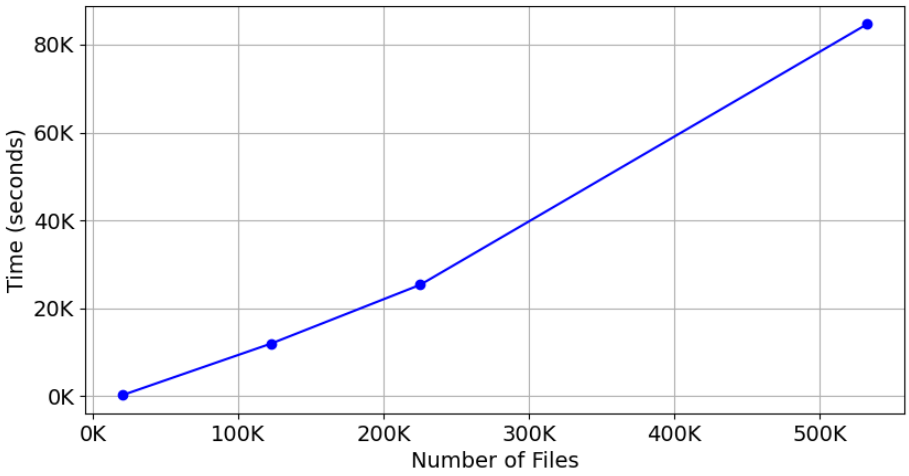


Figure 5.6: Phase 2 performance against various file quantities

5.5 Phase 1 Performance Analysis

As discussed in earlier chapters, Phase 1 is where pre-existing metadata attached to the files is being fetched processed and is inserted into the database. To analyze the performance of the proposed solution MITRA. We wrote a Naive approach to do the same steps keeping the file access and database management to the simplest and fastest form possible. Also, for this approach, we had to limit the code to a single file format to keep the performance of the naive approach optimum.

Initially, we observed the performance on a single node and observed the difference in performance. And we observed the overall time for MITRA was relatively high compared to the naive approach. Although in the MITRA approach the work was done by the client node itself similar to the naive

Task	Time for Naive	Time for MITRA
Import Libraries	0.29 s	0.27 s
Fetch header	337.23 s	166.81 s
Process Header	27.72 s	
Insert to database	311.59 s	723.01 s
Map Schema	-	1075.44 s
Total Time	680.88 s	1075.72 s

Table 5.1: Phase 1 performance for 1 worker node

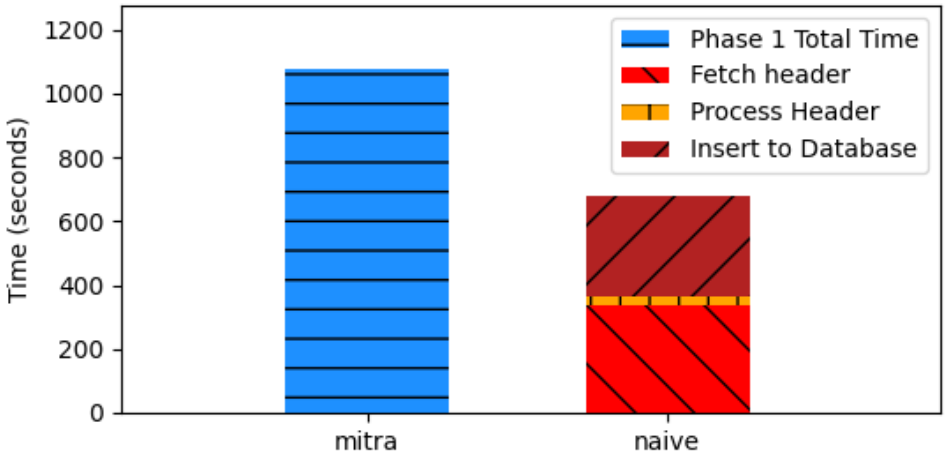


Figure 5.7: Phase 1 performance for 1 worker node

approach.

But, MITRA had many overheads to maintain multiple tables in the database, Maintain server-client communication for file access for scalability, Extension processing as MITRA supports multiple formats in a collection of datasets. Because of these overheads when we further observe the time spent on a deeper level it is observed that Naive is more efficient. Also, MITRA is

Task	Time for Naive	Time for MITRA
Import Libraries	0.21 s	0.17 s
Fetch header	151.73 s	74.31 s
Process Header	1.95 s	
Insert to database	22.4 s	75.66 s
Map Schema	-	88.44 s
Total Time	176.12 s	88.53 s

Table 5.2: Phase 1 performance for 18 worker nodes

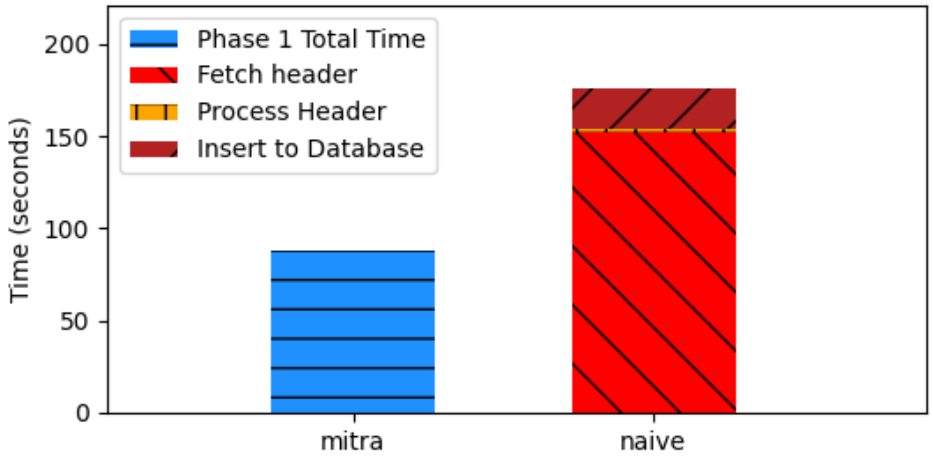


Figure 5.8: Phase 1 performance for 18 worker nodes

running more than one process at the same time for multiple tasks so the time taken by the longest process in parallel execution is taken to be the actual time for the performance.

After Analyzing performance for a single worker node. To understand the performance of MITRA on a scaled cluster environment. For this, we analyzed performance for 18 worker nodes. for MITRA approach it was possible to

distribute files easily using server allocation based on a heuristic tree. For the naive approach we manually Distributed files in the same 18 divisions and all 18 nodes were assigned unique files. When compared for 18 worker nodes and profiled to see which phase is creating overhead. The difference in total time taken for MITRA and naive approach is fetching the header time. The parallel multi-processing approach is optimizing time for header fetch, Database Management, and Mapping schema time as well. It can be said that the 18 worker node environment MITRA performs better compared to the naive approach based on Database management and Network file transfer time.

5.6 Phase 2 Performance Analysis

For Phase 2 performance analysis, We observed the performance of MITRA against the naive approach. For Phase 2 we initially observed the total time taken for both the naive and MITRA approach. Both the approaches are serial for phase 2 in which we can observe the time taken for each and every step for phase 2. As discussed earlier this phase is designed to extract metadata based on the content of the dataset and schema provided by scientific dataset users. This phase is heavily dependent on network content transfer as it fetches the contents of the files to process and extract metadata.

While analyzing the time taken for a single worker node. It can be clearly observed the time taken for file transfer in MITRA approach is more optimized compared to the naive approach. This improvement is observed because of the selective fetching of contents based on the schema mapping

Task	Time for Naive	Time for MITRA
Fetch Contents	410.76 s	171.87 s
Perform operations	12.27 s	15.03 s
Database Operations	192.09 s	322.02 s
Total Time	649.31 s	584.69 s

Table 5.3: Phase 2 performance for 1 worker node

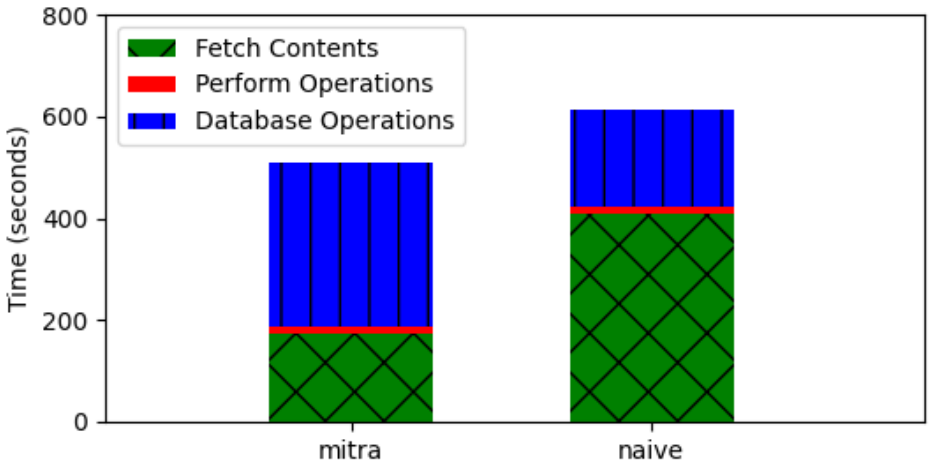


Figure 5.9: Phase 2 performance for 1 worker node

done additionally in the previous phase. Also, we can see a relative increase in database operation time as table updation in MITRA is a little complex compared to the naive approach.

After analyzing the performance for a single worker node. We scaled the experiment for 18 nodes in order to observe if there were any overheads caused in memory and network transfer with increased concurrency over file load and database management compared to the naive approach. for distribution in 18 node environment for Naive approach, we divided the datasets in 18 parts

Task	Time for Naive	Time for MITRA
Fetch Contents	293.65 s	220.73 s
Perform operations	1.97 s	0.36 s
Database Operations	24.22 s	7.74 s
Total Time	320.38 s	228.83 s

Table 5.4: Phase 2 performance for 18 worker nodes

as we did during phase 1 for a similar simulation of the experiment. In this experiment, we observed that the time taken for file transfer in MITRA is optimized compared to the naive approach. In MITRA for phase 2 when the schemas for metadata extraction are being processed we have already processed the initial schemas and labeled the files accordingly. This helps in accessing just the required files for phase 2 and not search every file for files containing the required set of values. For this experiment, we used the variable present in the 3/4th number of files. Table 5.8 we can observe that the time taken to fetch and process the selected files is less compared to fetching and processing all the files.

Also, we can observe that Database operation time is also optimized although we are processing 3/4th part of the given files the database operation is relatively much more optimized compared to the time taken for the naive approach for inserting the same data in the database.

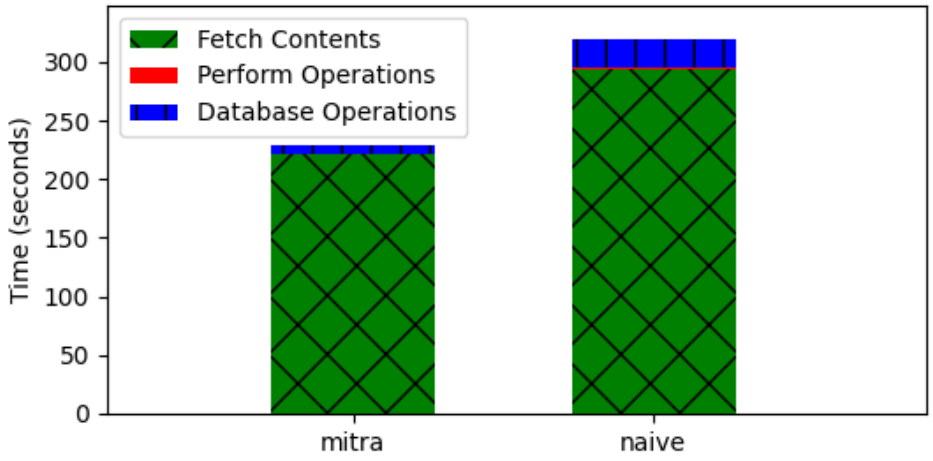


Figure 5.10: Phase 2 performance for 18 worker nodes

5.7 Overall Performance Analysis

After Observing the performance of MITRA and Naive approaches and measuring time taken for each task in the analysis of phase 1 and phase 2. When overall time is compared for both approaches for all the tasks of Phase 1 and Phase 2 we can observe that the overall time taken for a single node environment by MITRA is much more than the naive approach. As we discussed in the above observations, for phase 1 time taken in a single node environment is much more because of the overheads in phase 1. Although for phase 2 time taken is optimized in a single node environment.

As we discussed above for 18 node environment time taken for phase 1 and phase 2 for MITRA is much more optimized compared to the naive approach. Based on the observations and time division for the 18 node ex-

Task	Time for Naive	Time for MITRA
Import Libraries	0.29 s	0.27 s
Fetch header	337.23 s	166.81 s
Process Header	27.72 s	
Insert to database	311.59 s	723.01 s
Map Schema	-	1075.44 s
Fetch Contents	410.76 s	171.87 s
Perform operations	12.27 s	15.03 s
Database Operations	192.09 s	322.02 s
Total Time	1291.95 s	1584.64 s

Table 5.5: Overall performance for 1 worker node

Task	Time for Naive	Time for MITRA
Import Libraries	0.21 s	0.17 s
Fetch header	151.73 s	74.31 s
Process Header	1.95 s	
Insert to database	22.4 s	75.66 s
Map Schema	-	88.44 s
Fetch Contents	293.65 s	220.73 s
Perform operations	1.97 s	0.36 s
Database Operations	24.22 s	7.74 s
Total Time	496.50 s	317.36 s

Table 5.6: Overall performance for 18 worker nodes

periment it can be said that MITRA's file access management and database operation management is better for the scaled environment but for a single node environment, the overheads are quite high. This tells us that MITRA is more efficient in handling a high volume of datasets over a scaled distributed cluster.

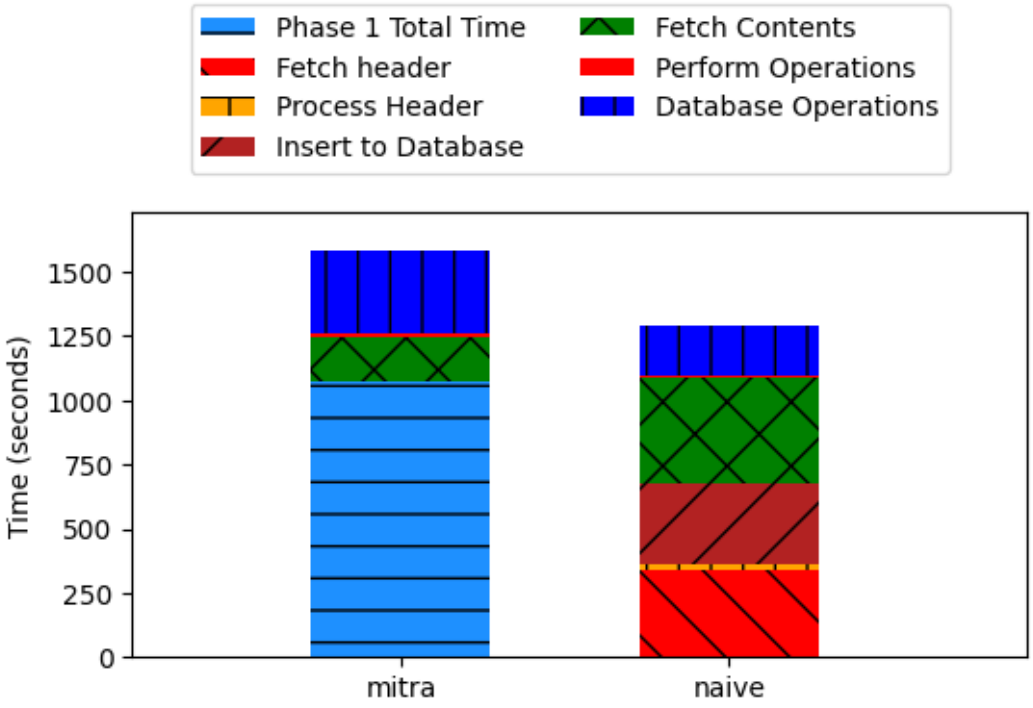


Figure 5.11: Overall time taken by 1 worker node

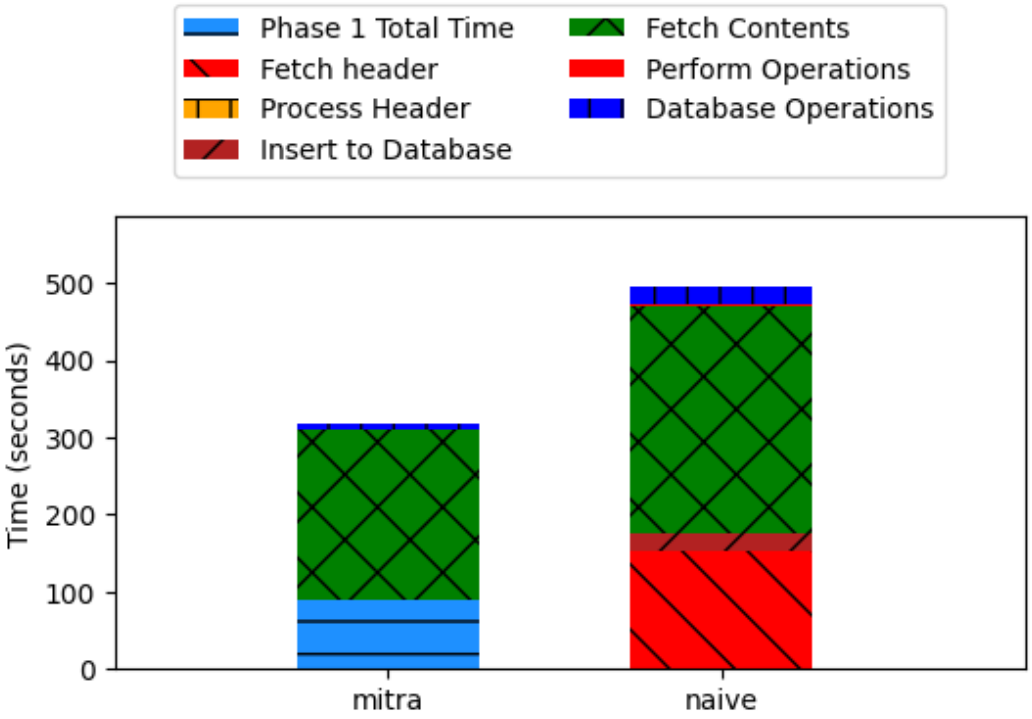


Figure 5.12: Overall time taken by 18 worker nodes

5.8 API Retrieval

Accessing the files from storage based on a specific set of values efficiently was one of the objectives for working on MITRA. After efficiently generating a database for given files we compared the time taken to retrieve a specific set of files based on the range of values of specific variables which may have been processed to be stored in the database or not.

As we can observe in the given table the values represented are of when if a set of files is requested based on the range of values a set of JSON values is returned following the condition. We compared the performance for database generated using MITRA approach and database generated using the Naive approach. For both the Databases we also tested the time taken to process when a certain variable is present in the database while processing for phase 2. We can see the time taken to return the list of paths for fetching is similar in both databases. The naive approach is a little efficient because it has one simple table. But when the files requested a certain variable that was not extracted during phase 2. The API request in such case would verify for all the files processing the metadata. In this case, MITRA performs more efficiently. As it does a schema-based dis-aggregation of files so when it has to access all the files we can shortlist the files containing the variable from the table storing the schemas. Whereas in the naive approach there is no schema-based classification which results in scanning all the given files.

	Field present in Database	Field not present in Database
MITRA DB	0.0552 s	157.3649 s
Naive DB	0.0496 s	205.7323 s

Table 5.7: Analysis for API value retrieval

5.9 File Tree Generation

As discussed in the above sections, we generate a Tree based on heuristic values containing directories as non-leaf nodes and files as leaf nodes. In this evaluation, we have analyzed the performance of time taken to generate file Tree in order to verify if it is causing any overheads with an increasing number of files and relative time taken. The table and chart below shows the best time taken for up to 1 million files. for generating a heuristic-based tree on multiple levels and chunkifying it based on the number of clients present.

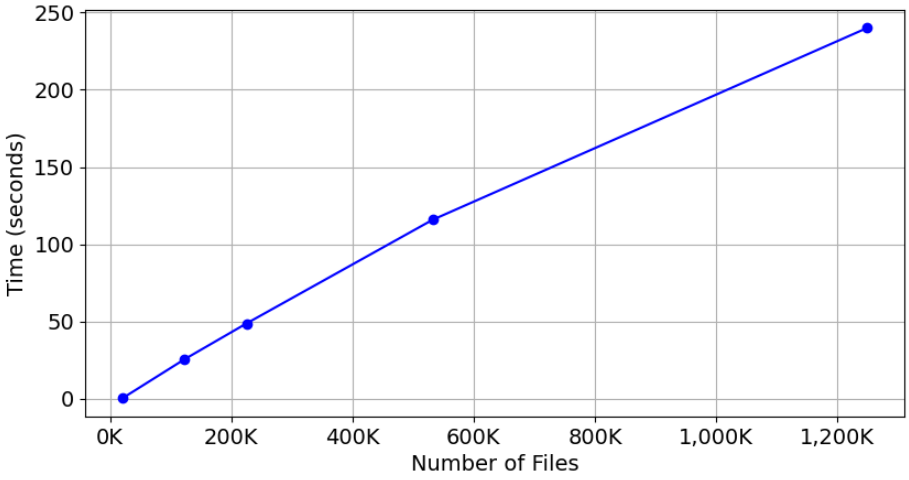


Figure 5.13: File tree generation

Chapter 6

Discussion

In order to describe a present system performing closer to the straightforward naive approach. We discovered multiple points where optimizations were possible in the approach to address the design goals. the significant optimizations that improved the performance of the system on a large scale are discussed below.

6.1 Heuristic File Tree generation

Heuristic File Tree is one of the core components in our system as it acts as a load balancer for the cluster and helps us in maximum utilization of the resources. As discussed in the design goals this file tree has a function to custom define the heuristic value to distribute load and to custom assign the value the tree needs to crawl through the entire file stack and create a tree data structure of it's own. After this tree is created it can be further distributed in chunks to be distributed among the client nodes to process the assigned files respectively in batches based on the capacity of each client node.

The initial approach we used to generate this file tree was a recursive approach in which the control of the program would crawl to every directory

and files in a depth-first search pattern and keep track of recursive traveling and construct the tree side by side. Once the tree generation was complete it would traverse the entire tree again in the bottom-up approach and assign heuristic values to every non-leaf node. After assigning the heuristic values based on the contents the chunks were generated for distribution requiring partial top-down traversal of the tree. The entire approach was sequentially performed by single-core as well. The python library used in this approach was os.listdir() to recursively process every directory. The overall time taken to generate a tree using this approach was significantly high which we realized later as we performed evaluations for bigger datasets.

To optimize this approach, We updated the approach from a recursive approach to an iterative approach. Instead of using os.listdir() library recursively we switched to os.scandir() and fetched a list of all existing files and directories. After generating the list we distributed the list to multiple cores present on the server to process the list and generate a tree. In above experiments as we were using the same server for database access, we used 2 cores less than the total number of cores present allowing one core for database access and one core for other system regular activities. So, if the cluster had 12 virtual cores we used 10 threads to process the file list on 10 virtual cores. When the list was distributed among multiple threads we had to set the tree as a shared data structure and as multiple threads might be accessing the tree the race conditions were also managed to avoid dirty reads and writes on the tree. The total count of files was verified in the database after every iteration of ex-

Task	20K files		2M files	
	os.listdir()	os.scandir()	os.listdir()	os.scandir()
Crawl through files	0.00828 s	0.00044 s	12.41237 s	0.00077 s
Generate list of files	0.22 s	0.053 s	23.46 s	14.68 s
Assign heuristic and generate tree	10.86172 s	0.22656 s	Unknown (48 hrs+)	455.31923 s
Total time taken	11.09 s	0.28 s	Unknown	470 s

Table 6.1: Analysis of two approach for file tree generation

periments from 400 to 2 Million files given to the system. To further improve this model because we were maintaining the race conditions on a shared data structure it was also possible to keep on updating the heuristic value of the tree while generation so two passes can be completed in the same pass. These optimizations helped us generate the desired tree much efficiently compared to the previous approach. After the generation of the tree with heuristic values, the second pass was to distribute it in chunks by partial top-down traversal which was the same as the earlier approach.

When we profiled both approaches for two corner datasets of 20 thousand files and 2 Million files the results were seen as follows after optimization.

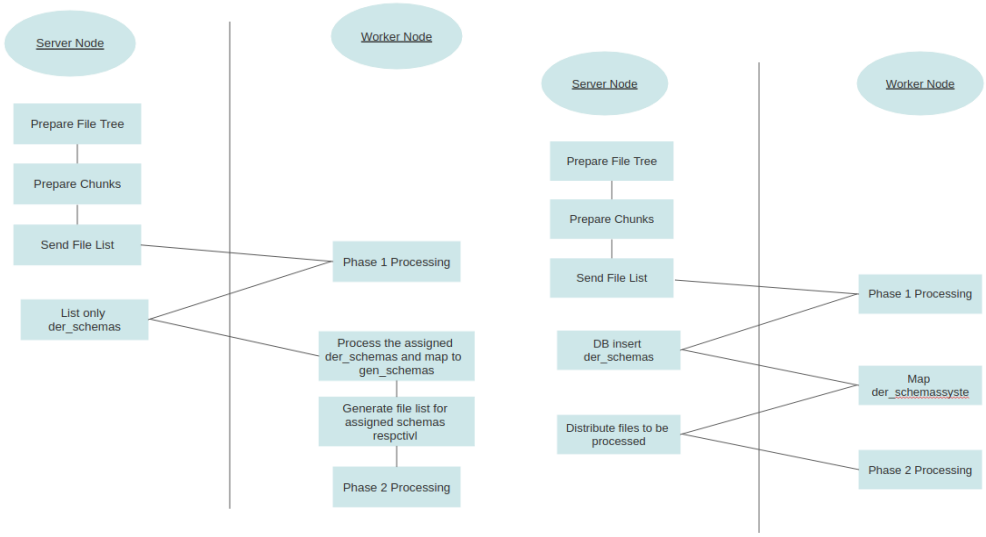
As we can see the performance optimization to crawl through the file system and generate a simple list of files shows a significant improvement and the iterative approach with parallelization is much more efficient compared to previous sequential processing of files.

6.2 Overall Flow of the System

Heuristic file tree generation was the first step for the system to start. After Optimizing the file tree generation we found a significant optimization on the overall control flow of the system. As discussed in the above chapters our system is based on Server-Client architecture. The division of tasks and communication is also important to optimally utilize the hardware and keep network traffic to a minimum.

As we can see in the initial flow after Phase 1 ends task of the server is just to read the der_schemas and distribute them to the clients. Since when this was designed the design was in nascent stages so der_schemas were assigned linearly to available clients and those client nodes were responsible to insert those schemas into the database, map them to respective gen_schema tables, generate file list for selected gen_schemas only and then fetch and process them. So, If there are 50K files to be processed, 18 client nodes and 2 der_schemas to be processed 2 clients are assigned one schema each, and the rest of the processing is left to only 2 client nodes respectively for files of each der_schema.

In the Later approach as we can see the distribution was made more evenly distributed. In this design also the server was responsible to read der_schemas but it was also responsible to insert them into the database and clients will map them to respective gen_schemas. After this server will generate the total list of files to be processed and distribute it evenly for available client nodes. So, If there are 50K files to be processed, 18 client nodes and 2 der_schemas to be processed server will generate and distribute-list of 50K



(a) Initial System Design (b) Final System Design

Figure 6.1: Overall system flow and server-client communication of MITRA.

files to be processed and all 18 nodes will be assigned files respectively. By this approach the We can better use the 18 client nodes by a little coordination from the server node.

6.3 Client node Optimization

While trying to understand the performance of the system for different datasets and different conditions We observed that the performance of our client node was taking a significant amount of time compared to the Naive approach node. When we profiled the performance for both the nodes we observed that our client had more tasks to perform in order to complete phase 1. As our node supported more than one file format such as .h5, .nc4, .csv, and

Approach	Fetch Header	Insert to Database	Map Schema	Total Time
MITRA (Serial)	34 s	39 s	64 s	137 s
MITRA (Parallelized)	20 s	70 s	74 s	74 s
Naive Approach	33 s	25 s	-	58 s

Table 6.2: Analysis of client node approaches

more at the same time whereas Naive supported only one file format so the processing pipeline and database management was more complex. Also, because of scalability feature coordination between server-client nodes and processing was creating heavy overheads. In order to optimize the performance of our client nodes, we switched to a multi-processing model. As discussed in Phase-1 discussion in earlier chapters. We have 3 primary tasks Fetching header from storage and reading metadata, Inserting metadata to the database, and mapping schema of each file in the chain of tables. For optimizing these 3 tasks in a single client node we have launched separate processes to handle each task executed in a pipeline model. There are Two queues present in between 3 processes for parallel pipeline execution as soon as every file is processed by one process. By this approach, we were able to see significant improvement in performance. The time taken by each stage of the multi-processed model is the total time taken to complete the process. As all 3 processes start at the same time for total time we have considered the longest-running process. In the table below we have profiled time taken by both approaches for dataset size of 1200 files of 800 GB for a single client node.

6.4 Database Management Optimization

As we are maintaining the metadata storage and index in a relational database management system. Database operations and management play a significant role in overall system performance. The Database storage and access through the system are also flexible to update table structure in real-time in case of new data field discovery or manipulation because of this processing and generating automated queries creates significant overhead. In order to optimize the overhead formed by these operations, we studied the time taken for our system to access the database and we observed the most of the time was taken by insert the operation carried during phase 1 to insert the metadata and file access info across the chain of tables. To improve the time taken to insert the metadata we switched the approach from using INSERT queries to COPY TO queries. These queries are used to insert bulk data from files to the database in the most efficient way possible. To ease the insertion we formed in memory tabular storage structure to hold the data that is to be inserted into the database using COPY TO queries. After switching to this approach we observed significant improvement in the time taken by the client node to insert the data into the database. Considering the time taken to handle and update in-memory tabular data structure and database interaction time COPY TO approach still performed better compared to the INSERT approach. In the table below we have analyzed the time taken by both approaches for a dataset size of 400 GB for a single client node.

No. of Files	INSERT Approach	COPY TO Approach
800	120 s	117 s
20480	1341 s	1060 s

Table 6.3: Analysis of time taken to add data to database

Chapter 7

Conclusion

In this thesis, we introduced MITRA, which is a distributed processing architecture to efficiently manage high volume scientific datasets for archival storage. This system enables traditional POSIX file systems to be more accessible and efficiently managed. We displayed that the performance of MITRA is better than the naive approach made to address the problem to generate metadata and manage a tabular index for easy access of files. Apart from that it also supports multiple file formats and flexible database schema management.

Our research has focused on providing a solution for scientific data user communities. Since in scientific research communities various file formats are used to store and exchange data and observations it becomes difficult to process that data later as contemporary systems do not provide support for effective indexing of data from all the file formats. MITRA has an interface comprising of four methods that can be implemented to allow processing of any custom file format for indexing its metadata and generate more metadata from the contents. MITRA is modeled on client-server architecture. Which enables utilization of sockets and multi-threading for effective management of message communications, data transfer, and concurrent processing over a

scaled distributed computing cluster. In our analysis, we observed that when scalability of MITRA is compared to the traditional naive approach, the strategies used by MITRA show significantly improved performance over traditional approaches. Also, when MITRA was tested against growing processing loads the execution time increased linearly with the increase in processing load. The heuristic tree-structured load balancer ensures that the composition of workload is balanced based on the computational strength of every client node. Batch processing allows every client node to receive batches of workload assignments based on the hardware configuration of every node. This ensures balanced distribution and optimum performance even on heterogeneous computation frames. Relational database index optimizes the lookup time to map file access information. MITRA has automatized database management for smooth insertion and updation of data to the database while performing ingestion of metadata. MITRA also manages to alter tables dynamically to allow strict Relational databases to be utilized flexibly. MITRA can also handle multi-level key-value paired metadata by converting it to a two-dimensional data structure and store it in the database with other metadata.

7.1 Future Work

Based on the results, remarks, and limitations of the research, there are still a few points that can be improved in the future to make the system overall more optimal and more user-friendly.

In terms of the failure of some components the ability of the system to

recover from the loss and reorganize itself to continue the execution is still a limitation. for instance if a client node drops then right now there is no way to relocate the file of that client to other nodes. similarly if one of the server's drop then the ability to replace the server and continuing the execution. Even the communication happening between server nodes and client nodes if one the message or data packet is corrupted then in place of crashing the ability to re-request and resend the information can be developed to make the system more reliable.

Besides, reliability optimization the API server also needs to accommodate multiple requests and simultaneous processing in order to serve multiple scientific data users for the same archival storage. Currently, the API server present is a single-process server that can at most utilize one core to respond to the query of the user. As well as certain in-memory databases can be used to respond swiftly with queries containing the data from previous queries saving the time to scan certain parts of the database.

Apart from the API server, we could also add real-time data analytics functions. For instance, while constructing the heuristic tree if we have past data of time take to process each file based on extension, size, depth of hierarchy, and other factors then for future runs the weight of the heuristics can be handled accordingly for more balanced distribution. The phase to metadata generation system can be made some more flexible to understand and process the metadata. For instance, if two fields are 'temp' and 'temperature' then instead of creating two separate columns it can merge them in the same col-

umn by adding an alias for both to access later. Similarly if for a specific a certain function is always performed like getting a range of coordinates, range of temperature values, mean of wind speed for majority .csv files containing those attributes. Then to making them variables of interest and generating those metadata for the rest of the .csv files as well containing those attributes.

Bibliography

[1] Arangodb, the multi-model database for graph and beyond. https://www.arangodb.com/.

[2] Ddap - articles and whitepapers. https://web.archive.org/web/20040111002537/http://www.ddap.org/resource_center/tiffit_position_paper.php. (Accessed on 07/04/2021).

[3] Understanding the kdf file. https://www.karma-lab.com/karmasoft/ko/files/Understanding_The_KDF_File.pdf. (Accessed on 07/04/2021).

[4] Joe Arnold. *Openstack swift: Using, administering, and developing for swift object storage.* " O'Reilly Media, Inc.", 2014.

[5] Murtha Baca. *Introduction to metadata.* Getty Publications, 2016.

[6] Chad Berkley, Matthew Jones, Jivka Bojilova, and Daniel Higgins. Metacat: a schema-independent xml database system. In *Proceedings Thirteenth International Conference on Scientific and Statistical Database Management. SSDBM 2001*, pages 171–179. IEEE, 2001.

[7] Saman Biookaghazadeh, Yiqi Xu, Shujia Zhou, and Ming Zhao. Enabling scientific data storage and processing on big-data systems. In *2015*

IEEE International Conference on Big Data (Big Data), pages 1978–1984. IEEE, 2015.

[8] Eric B Boyer, Matthew C Broomfield, and Terrell A Perrotti. Glusterfs one storage server to rule them all. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2012.

[9] Paul G Brown. Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 963–968, 2010.

[10] Joe B Buck, Noah Watkins, Jeff LeFevre, Kleoni Ioannidou, Carlos Maltzahn, Neoklis Polyzotis, and Scott Brandt. Scihadoop: Array-based query processing in hadoop. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2011.

[11] Arturo Crespo and Hector Garcia-Molina. Archival storage for digital libraries. In *Proceedings of the third ACM Conference on Digital libraries*, pages 69–78, 1998.

[12] Adam Crume, Joe Buck, Noah Watkins, Carlos Maltzahn, Scott Brandt, and Neoklis Polyzotis. Scihadoop semantic compression. 2012.

[13] Mohamed Dabees and J William Kamphuis. Nline: Efficient modeling of 3-d beach change. In *Coastal Engineering 2000*, pages 2700–2713. 2001.

[14] Frank Frank Edward Dabek. *A distributed hash table.* PhD thesis, Massachusetts Institute of Technology, 2005.

[15] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[16] Ananth Devulapalli and Pete Wyckoff. File creation strategies in a distributed metadata file system. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

[17] Erik Duval, Wayne Hodgins, Stuart Sutton, and Stuart L Weibel. Metadata principles and practicalities. *D-lib Magazine*, 8(4):1–10, 2002.

[18] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47, 2011.

[19] Felix Gessert, Steffen Friedrich, Wolfram Wingerath, Michael Schaarschmidt, and Norbert Ritter. Towards a scalable and unified rest api for cloud data stores. In *GI-Jahrestagung*, pages 723–734, 2014.

[20] HDF Group et al. Introduction to hdf5, 2018.

[21] Andreas Grünbacher. {POSIX} access control lists on linux. In *2003 {USENIX} Annual Technical Conference ({USENIX}{ATC} 03)*, 2003.

[22] Mahesh Lal. *Neo4j graph data modeling*. Packt Publishing Ltd, 2015.

[23] Margaret Lawson and Jay Lofstead. Using a robust metadata management system to accelerate scientific discovery at extreme scales. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, pages 13–23. IEEE, 2018.

[24] Margaret Lawson, Craig Ulmer, Shyamali Mukherjee, Gary Templet, Jay Lofstead, Scott Levy, Patrick Widener, and Todd Kordenbrock. Empress: extensible metadata provider for extreme-scale scientific simulations. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, pages 19–24, 2017.

[25] Jay Lofstead, Ivo Jimenez, Carlos Maltzahn, Quincey Koziol, John Bent, and Eric Barton. Daos and friends: a proposal for an exascale storage system. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 585–596. IEEE, 2016.

[26] E Laxmi Lydia, A Krishna Mohan, and M Ben Swarup. International journal of engineering sciences & research technology processing image files using sequence file in hadoop.

[27] Stephan Mäs, Daniel Henzen, Lars Bernard, Matthias Müller, Simon Jirka, and Ivo Senner. Generic schema descriptions for comma-separated

values files of environmental data. In *The 21th AGILE International Conference on Geographic Information Science*, 2018.

[28] Inc MongoDB. Mongodb. *URL https://www. mongodb. com/. Cited on (2014)*, 9, 2014.

[29] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and Dave Hitz. Nfs version 3: Design and implementation. In *USENIX Summer*, pages 137–152. Boston, MA, 1994.

[30] Jeffrey Pomerantz. *Metadata*. MIT Press, 2015.

[31] Ajinkya Prabhune, Hasebullah Ansari, Anil Keshav, Rainer Stotzka, Michael Gertz, and Jürgen Hesser. Metastore: A metadata framework for scientific data repositories. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3026–3035. IEEE, 2016.

[32] Kai Ren, Qing Zheng, Swapnil Patil, and Garth Gibson. Indexfs: Scaling file system metadata performance with stateless caching and bulk insertion. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 237–248, 2014.

[33] R. Rew and G. Davis. Netcdf: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82, 1990.

[34] Drew S Roselli, Jacob R Lorch, Thomas E Anderson, et al. A comparison of file system workloads. In *USENIX annual technical conference, general track*, pages 41–54, 2000.

[35] Frank B Schmuck and Roger L Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, 2002.

[36] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[37] Hyogi Sim, Awais Khan, Sudharshan S Vazhkudai, Seung-Hwan Lim, Ali R Butt, and Youngjae Kim. An integrated indexing and search service for distributed file systems. *IEEE Transactions on Parallel and Distributed Systems*, 31(10):2375–2391, 2020.

[38] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 20:24, 2011.

[39] Houjun Tang, Suren Byna, Bin Dong, Jialin Liu, and Quincey Koziol. Someta: Scalable object-centric metadata management for high performance computing. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 359–369. IEEE, 2017.

[40] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2011.

[41] Chandramohan A Thekkath, Timothy Mann, and Edward K Lee. Frangipani: A scalable distributed file system. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 224–237, 1997.

[42] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, 2006.

[43] Brian S White, Michael Walker, Marty Humphrey, and Andrew S Grimshaw. Legionfs: A secure and scalable file system supporting cross-domain high-performance applications. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, pages 59–59, 2001.

[44] Qing Zheng, Kai Ren, and Garth Gibson. Batchfs: Scaling the file system control plane with client-funded metadata servers. In *2014 9th Parallel Data Storage Workshop*, pages 1–6. IEEE, 2014.

Sarthak Thakkar

Curriculum Vitae

Education

2019–2021 **Masters of Sciences**, *Rochester Institute of Technology*, Rochester, *GPA – 3.48*.
Specialized in Bigdata Analytics and Management

2015–2019 **Bachelor of Technology**, *CHARUSAT University*, India, *GPA – 8.60*.
Graduated with Distinction

Masters Thesis

Title *MITRA: Robust Architecture for Distributed Metadata Indexing*

Supervisors Dr. M. Mustafa Rafique

Description Applied a novel scalable metadata indexing architecture over distributed network with unique capabilities of metadata generation for richer metadata index enriched with improved ingestion performance and file accessibility for Archival storages containing scientific datasets.

Experience

2020–2021 **Graduate Research Assistant**, Rochester Institute of Technology, Rochester.
Developed a state of the art scalable architecture for analysing and managing various scietific format datasets for exascale archive storages in Collaboration with Oak Ridge National Labs.

Spring 2019 **Machine Learning Engineer**, Perfect and Complete Solutions, Pune.
Developed a Computer vision service over fog architecture using Caffe model and PyTorch to monitor room usage with Demographics, Track map, footfall and presence using RaspberryPI, RSTP remote cameras.

Summer 2018 **Summer Intern**, Streebo Inc., Ahmedabad.
Developed a cross platform telecommunication dashboard to monitor services and user management.Integrated payment gateway and ACID transaction management for wallet provided by service provider.

Spring 2016 **Trainee**, Excelsior Technologies, Ahmedabad.
Developed an e-governance portal for my university on Laravel framework using OAuth2, Ajax, Jquery, HTML5/CSS3.Portal supporting authentication, user management, content management and academic activities.