Hadoop MapReduce

Anonymous

ABSTRACT

Data is continually expanding at an exponential rate, and lack of physical resources to tackle the needs lead to different issues related to scalability, fault tolerance, flexibility. Currently, the typical size of a consumer hard disk ranges from one to fifteen terabytes with rotational speeds in the range of 5400-10000 rpm. Transfer speeds of 500MB/s would take around half an hour to read the entire data, which brings the need for parallel processing and distributed storage for improving computational time. Storing data in a distributed manner introduces the risk of data loss due to hardware failure. The traditional way of solving this issue is to implement RAID, which stores multiple copies of data. Hadoop distributed file system (HDFS) uses a different mechanism called just a bunch of disks (JBOD) architecture, which sequentially connects the disks. Another issue with most data analysis tasks is to merge data from different sources to generate results. By leveraging the advantages of Hadoop Map-reduce, we discuss the information about storing data in HDFS, indexing, processing, and optimization of query, transaction management, and an application that compares the performance of querying data using map-reduce and RDBMS.

1. INTRODUCTION

The term NoSQL databases have more than one interpretation. However, the interpretation most commonly used is 'Not Only SQL.' NoSQL databases are a type of database that provides additional functionalities over traditional relational databases so that companies can accommodate the ever-growing and changing requirements of their modern applications. These databases are best suited for applications that produce an enormous amount of data rapidly and of different data types like structured, semi-structured, or polymorphic data. The primary motivation behind the development of NoSQL databases was to provide flexibility and adaptability to match the pace at which data requirements change in the modern era while maintaining a high level of performance. Internet of Things (IoT) was another factor that pushed the development of database management techniques like NoSQL because of the need to support different gadgets producing different data structures. The key features of a NoSQL are being schema independent, having the ability to be highly distributable, being highly scalable, and being non-relational.

1.1 NoSQL databases vs RDBMS

Traditions RDBMS techniques were developed in an era before the Internet and Big Data. So it is natural that they have sometimes fallen short to the needs of modern applications that generate data at such a high rate. These limitations of RDBMS, such as scalability and inability to handle different data types, paved the way for the development of NoSQL databases, which were developed to cater to the needs of these data-heavy applications. Although NoSQL systems cannot replace RDBMS, these are some of the advantages they have: -

- Ability to handle change: - As NoSQL systems are free of schema, any changes incorporated in the data or the data structure is handled without having to rewrite the schema. This flexibility, in turn, allows for faster deployment of changes or updates. Being schema-independent also allows the application models to define the data models and thus allows faster development.

- Support for different functionalities: - NoSQL is of

different categories like Key-Value Based stores, Column Based stores, Graph-based stores, and document-oriented stores, which should be chosen based on the application requirements. Each of these categories provides the ability to handle different data types.

- Support for multiple data types: - NoSQL databases can support different data types by using the different functionalities provided. For example, Interrelated information can be stored using graphs stores.
- High scalability: - NoSQL databases provide are capable of being scaled horizontally.

1.2 Hadoop History

The Hadoop was initially a subproject of a bigger project of creating a web search engine called Apache Nutch. For solving the scalability issues in their project, Doug Cutting and some other Nutch developers built their own distributed file system (NDFS) and integrated the MapReduce framework. Hadoop was able to beat Google's MapReduce to achieve the fastest performance in sorting 1000 gigabytes in less than 62 seconds. [13]

1.3 HDFS Architecture

HDFS is a distributed system Distributed file system stores files according to the client and server architecture. Multiple servers store the same data accessed by the user. The storing and retrieving is done in the same way as a hierarchical system. HDFS system is developed to run efficiently on commodity hardware. Commodity hardware is the offshelve device, which is a general-purpose computer, inexpensive and can be used for different functions, unlike specific computer device. HDFS architecture consists of two types of nodes: NameNode and DataNode.

HDFS is a master/slave architecture in which namenode acts as a server, and datanode acts as a slave. Namenode stores all the information of filesystem metadata and provides access to the data according to the privilege. Namenode is the point of contact for the clients to access files on HDFS. Namenode is responsible for generating a request according to the requirement and retrieving the file from the datanode. Namenode stores all the information about file systems and the location of the file on the datanodes. Namenode is the only node that stores information about each file, which means it acts as a centerpiece in the whole system. The efficiency of the namenode for retrieving the files is critical. Usually, more RAMs and memory are provided to the namenode for efficient performance. Datanode acts as the slave node. Datanode stores data blocks, which are size around 128MB. Datanode is responsible for serving read and write requests of the namenode. Datanode

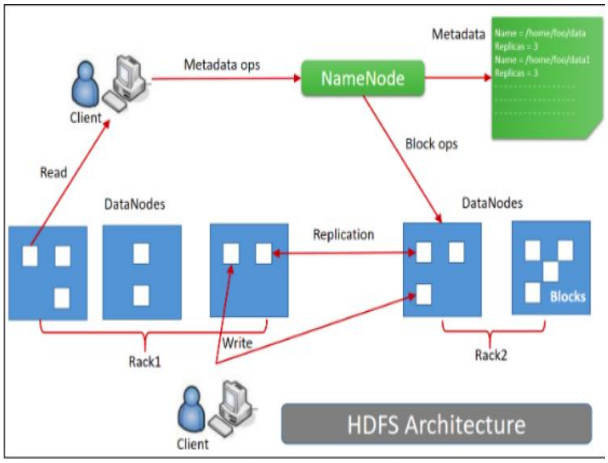

Figure 1: Hadoop Architecture [1]

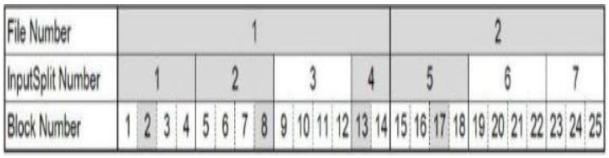

Figure 2: Indexing [2]

performs different types of functions like deleting a block, replacing a block, communicating with the data nodes according to namenode's request. Namenode is responsible for keeping track of each block of data and locations of the datanodes. Namenode is a single point of failure in the whole system.[9][6][5]

The paper is organized as follows. Section 2 presents the data storage and indexing. Section 3 provides query processing and optimization. Section 4 introduces the transaction management and security support. Section 5 discusses the NoSQL database application. Section 6 comprises the final remarks. Section 7 describes the work distribution of this project. Section 8 lists the references.

2. DATA STORAGE AND INDEXING

Data storage in Hadoop is performed by the Hadoop Distributed File System (HDFS). Each file is divided into different blocks (default size 128MB), which are distributed across the datanodes. By default, the factor of replication is three which means three copies of the data are available across different nodes. Each file can have a different configuration for the block size and the replication factor, which provides a significant advantage in case of the data node crashes or fails. The namenode received periodic signals from the datanode. Based on the periodic signals, namenode decides if the data block is working correctly or not. After the failure of datanode, namenode starts replicating the data blocks that were

available on the crashed data node. If one the datanode fails, still the data is not lost because it is already available on the different datanode. HDFS only allows writing and deleting of the data. It is not possible to update data because the data is being replicated on the different data nodes. If the modification is done on the data, then it should be applied to all other data nodes. This decreases the throughput of the system.

Relational Database Management System stores structured data in table format. Hadoop is not a database, but it is a file system that allows storing and processing large amounts of heterogeneous data like textfiles, pdfs, images, audio files across the data nodes. RDBMS system gives better performance for the only for smaller datasets. It is performance time to start degrading as the data gets large. With the help of MapReduce, it processes large amounts of data very efficiently. SQL is used in RDBMS for data retrieval and modification. Hadoop provides similar query language which Hadoop Query Language to allow user works with the data in a similar way as RDBMS. Apache Hive access the HDFS for storing the data. It creates a new file on the HDFS for a new database. Hadoop provides horizontal scaling, which allows the client to store unlimited amounts of data on nodes.

HDFS stores data in files, indexing is different from the indexing of RDBMS. MapReduce job is used for scanning the data. The indexing in the HDFS is expensive for regenerating the index of changing data. The index files are stored on the file system like a standard file. Three types of indexing are possible: Indexing on File, Indexing Based on block generated using Input Split, and indexing on the block.

- File Based Indexing: File-based indexing uses a key/value pair to index data. Keys store data, which is used for retrieval and value, stores the name of the file which contains this data. When retrieval is performed for the data, it returns the list of all files which contain the key. It is similar to the full scan. So, if the data is divided into three files, this needs to perform a full scan of files that matches the given key. As shown in the figure, indexing on file requires a full scan of the first page.

- Input Split Indexing: Input Split indexing divides each file into different blocks according to the split value. Rather than storing the file URI as the value in the key/value structure, it stores the list of blocks generated based on the given input split. It reduces the search space very effectively. As shown in the above diagram, instead of doing a full scan on the first file, it performs a search on only the first, second, and fourth block. Input Split indexing improves efficiency when data is extensive.

- Block Based Indexing: As the data is stored across as data blocks on multiple datanodes, it is possible to generate an index on the blocks. In this type of indexing, a list of data blocks is stored in the value in key/value pairs. The number of blocks increases exponentially, and it is difficult to maintain because its value keeps changing as the data added or removed.[11][7][8]

3. QUERY PROCESSING AND OPTIMIZATION

Query processing involves breaking down the high-level language queries into a machine-understandable format for generating query results. Different components are responsible for processing a user query. The user provides a query to the database, and this query gets tokenized and evaluated on their correctness. These tokens are converted to a tree of operators, and different query plans are generated. There are two ways of optimizing the query plans. The first one is to push down the selection and projection operations and use sort-merge join or block nested loops join. The other is to use indexing and pipelining to perform selection and projection on-the-fly. The best plan is selected and passed to the command processor, which fetches the query results from the database.

Apache Hadoop uses Hive query language (HQL) that is semantically similar to SQL but provides many benefits on top of SQL. The HQL provides enhanced performance because it supports different frameworks like MapReduce, Spark, and Tez. The primary data structures of this language are tables, partitions, and buckets.

1. Tables are the main directories of the HDFS and divided into two types: internal and external. The internal tables are stored in the warehouse directory and act as the main copy of data. If the internal table is dropped, the data gets deleted from the file system. On the other hand, the external table is like a clone of an internal table which is stored in the location properties. Using external tables is a safe choice, as deleting external tables does not result in the deletion of the original data in the database.

2. Partitions are smaller portions of the table, which are used for faster querying that includes only selected columns.

3. Buckets are required when we want to subdivide the data of partitions. A column from the partition is chosen, and hashing is performed to segregate them into buckets specified by the user to avoid nested partitions.

These data structures are helpful in data sampling and joins during the mapper function. Views hide the complicated part of a query like joins, subqueries, and filters, and the data does not get materialized.

The MapReduce is a Hadoop builtin framework that performs two major tasks, namely Map and Reduce. The map breaks down the input data into key-value pairs. The reduce function takes sub-parts of the output from the mapper function as input to generate the results. The MapReduce framework provides one of the best solutions for scalability because once an application is represented in the mapper and reducer format, we can use the application in thousands of machines. Also, if a user queries a database, the data node that is physically closest to the user gets processed to return the result.

The first task of processing is the task of mapping. The Mapper process each input record and processes it. The output of the mapping task is a collection of corresponding key-value pairs generated for each input record. As Mapper only understands key-value data, pre-processing of the data is required to make the data compatible for the Mapper. This pre-processed key-value data gets passed as an input to the Mapper. Now that the input is ready, we can discuss the complete process that occurs in the Mapper, in detail.

The workflow of the Mapper function:-

- InputFormat - Initially, the input text files are passed into the InputFormat function, which divides the files into chunks, and these chunks are then passed on to the InputSpilt function.
- InputSpilt - This is the representation of data in a logical format, and it describes a single map task in the MapReduce program. The output of InputSpilt gets passed as an input to the RecordReader.
- RecordReader - RecordReader converts the data into key-value pairs to make the data compatible for passing it as an input to the Mapper function. RecordReader communicates with the InputSplit in order to achieve this.
- Mapper - The Mapper takes the key-value pairs generated by the RecordReader as an input and processes them in order to generate the output. The output of the Mapper is also key-value pairs. However, these are called intermediate key-value pairs, which gets passed into the Reducer function.[12]

The workflow of the Reducer function:-

The Reducer is the second and final phase in the MapReduce program. The Reducer works with the output of the Mapper function and processes it into to generate the final output of MapReduce. Traditionally, in the Reducer phase, the aggregation/summation sort of computation is performed. All the reducers run in parallel as they work independently on the outputs of mappers for different individual map tasks. The number of reducers is user-defined.

Phases in Reducer:- There is a total of three phases in the Reducer function: the shuffle phase, the sorting phase, and the reduce phase.

- Shuffle Phase:- The main objective of the shuffle phase is to extract the parts of the mapper outputs that are important for the reducer phase. It works on the mapper outputs that are sorted by the sorting phase of the Reducer.
- Sorting Phase:- This phase happens parallelly with the shuffle phase. The sorting phase sorts the output of the mapper functions based on how similar the keys are. The shuffle phase uses these sorted outputs and processes them further.
- Reduce phase:- The final task of the reduce phase involves the aggregation/summation of the key-value pairs. The outputs are then collected using the OutputCollector.collect() method and generates the final output for the MapReduce program.

Steps to optimize the performance of MapReduce : -

- Use map joins: When dealing with a large number of data, using traditional joins that are used in SQL would drastically slow down performance. Thus, while performing joins in Hive, it is always better to use alternate methods of joins like Map join, bucket joins, Skew join, and so on, which would optimize the hive performance.
- Use partition: Make use of the partition concept of Hive. The partitioning concept divides the data into smaller chunks based on a specified column. Selecting how to partition the data is a crucial decision. Nevertheless, once partitioned, the data chunks can be processed in parallel, which would significantly improve the hive performance.
- Allow Parallel Execution: As discussed above, some phases of the MapReduce program can occur concurrently. For example, the shuffling and sorting phase of the Reduction phase can occur parallelly. By allowing the parallel execution feature, we can significantly improve the performance of Hive.
- Cost-Based Optimization: This Hive feature allows to place a check on the cost of queries and optimizes the queries based on cost.

4. TRANSACTION MANAGEMENT AND SUPPORT

Transaction Management is considered to be one of the essential parts of the database systems, in terms of consistency, reliability, and concurrency of database systems. The main objective of the database systems is to keep the data consistent with faster access and concurrency in the systems. As we know, relational database works well with a low volume of data, and the NoSQL databases work on a large volume of data which can be structured as well as unstructured.

In MapReduce large volume of data is processed in distributed form and processed parallelly, and data is stored in the HDFS file system, which is a distributed storage system compared to the storage of data in the single local file system by the RDBMS.

Now going into depth in the transactional properties of both relational and NoSQL databases, let us cover some critical points. Firstly, relational databases work on the principle of ACID PROPERTIES, which is an acronym for Atomicity, Consistency, Isolation, and Durability. All these properties are discussed below:

- Atomicity: It means that the transaction should be complete, or it should be aborted or rollbacked completely.
- Consistency: This property is fundamental in databases as the databases should provide consistent data.
- Isolation: This property defines a goal of the database systems that one transaction should not affect the other transactions occurring in the database system.
- Durability: This property of database systems defines that the transactions which are committed should remain committed, and there should not be any disappearing of the committed data.

Relational databases like Oracle, MySQL support these relational properties, whereas they are not supported by the transactional database used by Hadoop Framework, which is a Hive database because the given framework does not support Begin, Commit, and Rollback. Because ACID properties are very restrictive and care a lot for data security, we use the BASE consistency model in NoSQL. The BASE model stands for Basically Available, Soft state and Eventual Consistency.

- Basically Available - means the availability of the database. If a single node fails, a part of the data may be unavailable, but that does not affect the entire data layer.

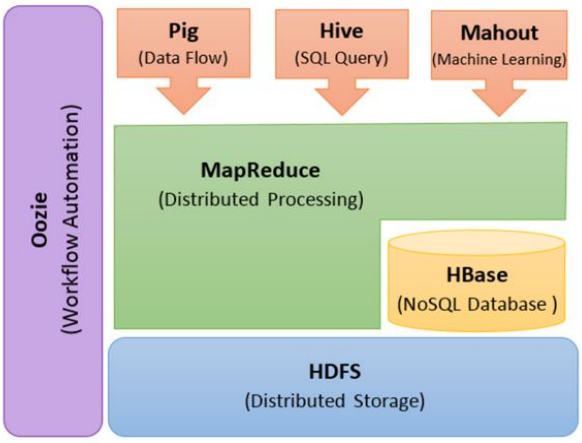

Figure 3: HDFS Structure [3]

- Soft State - This concept indicates that the database state may change over time without ever requiring an input.
- Eventual Consistency - This concept says that given time, an update to the database would reach every server. Nevertheless, the database does not wait for every server to be available. Instead, it starts updating the servers as soon as the change is made.

Below are some problems which are faced in the relational databases but not found to be in Hadoop MapReduce Framework:

DIRTY READ PROBLEM: Dirty Read is a problem caused due to the parallel processing of the transactions, and one transaction affects the value of the object the other transaction will be working on. This is a problem in the database systems as firstly, it fails the Isolation property of the database systems as each transaction is independent and is not aware of other transactions happening around, and so ideally, one transaction should not affect others. In this case, the write operation is performed by Transaction T1 and before commit of transaction T1, any other transaction T2 reads the value, it starts working on the value which is not committed to the main memory and if any failure occurs then the value in the database is not consistent. Also, if further T2 commits and T1 updates the value of the object further, it leads to data inconsistency.

UNREPEATABLE READ PROBLEM: Unrepeatable Read is the problem that causes one transaction to read different values of the same object at two different times, even when the reading transaction does not update the transaction. This defies the isolation property and so also affects the consistency and concurrency of the data.

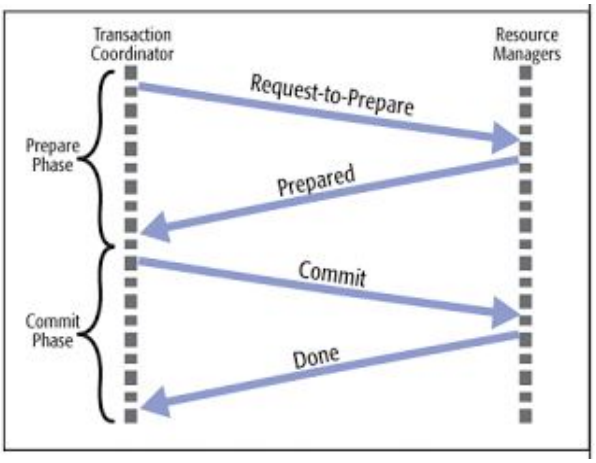

Figure 4: Two-phase Commit Protocol [4]

PHANTOM READ PROBLEM: It is a problem in the database where a transaction is trying to access an object deleted from the database by some transaction as it assumes that the object resides in the database as it was read previously by the same transaction.

LOST UPDATE PROBLEM: In this problem, the object value is read by some transaction T1 in from the main memory and is further updated by the same transaction, and some other transaction working in parallel tries to update the value of the same object without reading it. Further, the Transaction T1 commits the transaction without looking at the updated value made by another transaction, and so the update is lost.

2 PHASE LOCKING PROTOCOL: In relational databases, the two-phase locking protocol is used to prevent the deadlock condition. In the relational database, the above-discussed problem can occur, and so can be handled by this rule. In this protocol, the transactions which are being processed in parallel are allowed to lock an object on which they are working so that other transaction cannot access that object, also transactions are allowed only to lock in the growing phase and once the unlock on any object is performed then no transactions are further allowed to lock any object. In Hadoop MapReduce, the standard concept of avoidance, ignorance and prevention is used as in distributed systems and YARN Fair Scheduler is also used for resource management.[14]

2 PHASE COMMIT PROTOCOL:
In Hadoop MapReduce, 2 - Phase Locking Protocol (2PL) is used for prevention of deadlock by locking or proper sharing of resources, the 2 Phase Commit Protocol (2PC) is used for the inconsistent updating of data on the clusters or the DataNode, so that data on every node remains the same in case of the replicas, so this protocol ensures Atomicity as the process is carried out entirely or is aborted. In this protocol, there is a concept of Transaction Coordinator and Resource Managers/Participants. In this case, they are NameNode and DataNodes, respectively. The protocol works as follows; the NameNode asks for the commit votes from the respective data nodes and meanwhiles wait for the response. DataNode responds whether it is ready to commit or not and casts its response, which is collected by the NameNode. The NameNode will commit the transaction when a response from all the nodes is yes to commit else NameNode will abort the transaction so that the data in every data node replica remains consistent. Further, another reason for aborting a transaction is if the NameNode does not receive any response from any datanode. Datanodes can perform abort if it is in the initial state and does not receive any request for a vote from the NameNode or when it is ready to state by sending the response to the NameNode and is waiting for a response from the NameNode whether to global commit or global rollback.[10]

CRASH and RECOVERY: As we know, Hadoop uses a distributed file system to perform data operations, having metadata stored in NameNode and actual data in DataNode. If there happens to be a crash of DataNode, the DataNode stops sending heartbeat signals to the NameNode. This means that DataNode has failed. To handle the data loss in this case, the DataNode is replicated the distributed file system by default three times, and so the failed DataNode is replaced with the replicated DataNode. However, the main problem arises when there is a single point of failure, which is the failure of NameNode in this case. As NameNode assigns the work to the DataNode and stores the metadata, failure of this node can cause a massive impact on the reliability of the storage system, and the NameNode has to be brought back again manually. This was a problem in the Hadoop 1.0 and was addressed in Hadoop 2.0 with the concept of HDFS High Availability. In this concept, the NameNode is configured on two separate machines where one will be active, and the other will be passive, and till then, the passive machine acts as a slave and replaces the failed NameNode in case of failure of the working NameNode.

5. NOSQL DATABASE APPLICATION

For running Hadoop MapReduce, we installed a Cloudera VM in the Oracle Virtual Box virtualization software. This virtual machine is recommended to have at least 10gb

of RAM assigned to it and 4 CPU cores. The benefit of installing this virtual machine is that it provides the HDFS instances as well as all the services in the Hadoop framework, which can be installed separately. The components include the HDFS file system, Hive, Impala, MapReduce, PigLatin. This operating system also provides terminal support as well, where we do not have to install anything further.

We took the city database as an input, which has columns like ID, Name, CountryCode, District, Population. We stored this database as a CSV input in the HDFS folder. We implemented java code, which takes SQL queries as an input and generates output using MapReduce Framework. We have implemented the following type of query:

- Select * from table
- Select column_names from table
- Select column_names from a table where [conditions]
- Select column_names from a table where [conditions] group by field
- Select count(*) from table where [conditions]
- Select count(*) from a table where [conditions] group by field
- Select field, MAX(column_name) from table where [conditions] group by field
- Select field, MIN(column_name) from table where [conditions] group by field
- Select field, SUM(column_name) from table where [conditions] group by field

The output of these queries is generated in the text file on HDFS. The motive behind this project was to compare the performance of Relational Database Management System Hadoop and MapReduce for SQL queries as MapReduce works on a principle of distributed systems. We implemented SQL queries on both relational databases as well as Hadoop Map Reduce.

6. FINAL REMARKS

Even though the Hadoop MapReduce works on distributed systems principle, when we tested the SQL Queries by running it on both RDBMS and MapReduce, we found out that the performance of the MapReduce is prolonged compared to the relational databases. Also, the indexing in RDBMS improves the performance of query processing. On further research, we found out that the performance of MapReduce is better when the volume of data is large i.e., in PetaBytes.

Future Work can be done by running a project on high configuration servers and large datasets. More types of queries like sub-queries, joins, unions, and so on can be implemented using Hadoop MapReduce.

7. APPENDIX

The work distribution is as follows:

1. Abstract,Introduction - Nihal Parchand
2. Nosql vs RDBMS - Nihal Parchand
3. Hadoop history,Architecture - Akash Desai
4. Data Storage and Indexing - Akash Desai
5. Query Processing and Optimization - Nihal Parchand
6. Transaction Management and Support - Viraj Chaudhari
7. NoSQL Database Application - Akash Desai
8. Final Remarks - Viraj Chaudhari
9. Project implementation - Everyone
10. Presentation - Everyone

8. REFERENCES

[1] `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`.

[2] `https://www.codeproject.com/Articles/587383/InputSplit-Indexing-on-Mapreduce`.

[3] `https://www.codeproject.com/Articles/587383/InputSplit-Indexing-on-Mapreduce`.

[4] `https://medium.com/@balrajasubbiah/consensus-two-phase-and-three-phase-commits-4e35c1a435a`

[5] Hdfs architecture.

[6] Hdfs architecture guide.

[7] Index processing in hadoop.

[8] Indexing in hadoop.

[9] An introduction to hdfs.

[10] Two phase commit in mapreduce two (mr2).

[11] D. . gold badges55 silver badges1010 bronze badges, user8485334user8485334 4122 bronze badges, and V. S. S. . silver badges77 bronze badges. Indexing process in hadoop, May 1966.

[12] D. Team. Hadoop mapper - 4 steps learning to mapreduce mapper, Nov 2018.

[13] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition, 2015.

[14] Z. Yang, J. Bhimani, Y. Yao, C.-H. Lin, J. Wang, N. Mi, and B. Sheng. Autoadmin: Automatic and dynamic resource reservation admission control in hadoop yarn clusters. *Scalable Computing: Practice and Experience*, 19, 03 2018.