Tribhuvan University

Institute of Engineering

Pulchowk Campus

Assignment Report

Elective II: Big Data

Submitted By:

Prashant Karn

076BEI026 (Group B)

Submitted To:

Department of Electronics and Computer Engineering

Assignment date: 10/02/2024

Submission date: 02/03/2024

Assignment for Elective II: Big Data

Question summary:

1. Role of distributed systems in big data processing.
2. Current trend in big data analytics
3. Real world problem modeling in functional style
4. With example explain map reduce technique
5. Explain architecture and working of cassandra mongodb and HBase
6. With example explain distributed searching with elastic search
7. Explain Hadoop
    a. Installation on different environment
    b. Architecture
    c. Working
    d. Map reduce with hadoop with example and code

Question answers:

1. Role of distributed systems in big data processing

Distributed systems play a crucial role in big data processing. This is simply a result of the huge amounts of data in the scope of big data technologies. The idea is to make use of existing or easily accessible hardware to handle such huge amounts of data instead of using the costly route of building a single capable machine. Distributed systems are instead a collection of computers interconnected together, working together to achieve a common goal. These are well paired with big data processing because they offer the following advantages in the corresponding aspect of big data:

- **Parallel Processing**
  The vast majority of useful processing tasks to be done with real life examples of such large amounts of data is generally able to be parallelised. For example, if there are a lot of images that need to be scanned through to search for a particular face using facial recognition, it doesn’t have to be done sequentially, and different subsets of the data can be searched through by different computers. Distributed systems are ideal for this

- **Scalability**
  A custom machine capable of handling big data may not be scalable, either due to space constraints, heat constraints, power availability, hardware incompatibility, costs, etc. Distributed systems are built on the foundation that each component or node is modular and there are systems to handle their modularity, which makes it easy to add more nodes or combine multiple systems to vastly and easily increase the amount of data the system can handle

- **Data replication and redundancy**
  Distributed systems offer the advantage of having multiple copies of data in different locations. Since nodes do not share physical vulnerability, this can prevent data loss.

- **Data subsetting and partitioning**
  Distributed systems are designed to distribute data, for which the data needs to be split into subsets to be processed separately and in parallel by different nodes.

- **Resource management**
  Big data technologies require careful resource management in order to maximise the efficiency at which the data is being processed. There are technologies for this built into distributed technologies, such as YARN (Yet Another Resource Negotiator) for Apache Hadoop. Apart from this, there are even lower level technologies such as MapReduce that can distribute processing of data into many nodes and efficiently recombine the results

- **Real-time Processing**
  For applications such as surveillance where data is always actively being added to the system and need to be processed quickly to find valuable results, distributed systems have technologies such as Apache Storm that can handle this high data velocity by making good and efficient use of the large number of nodes

As a conclusion, distributed systems contain abilities and technologies that play a crucial role into making big data processing feasible and cost effective, and have been developed over the years to provide these features easily and efficiently.

2. Current Trend in big data analytics

Data is becoming a crucial part in each and every fields. Every field generates huge amount of data that needs to be processed to get valuable information. Each day we are producing more data in 2 days than decades of history.

Some of the current trend in big data analytics are:

Automated Machine Learning (AutoML):
The growing complexity of machine learning models has led to increased interest in AutoML, where machine learning processes, including feature engineering, model selection, and hyperparameter tuning, are automated to make machine learning more accessible

Responsible and Smarter AI:
Integrating ethical considerations into AI development for responsible and intelligent decision-making.

Predictive Analytics:
Utilizing advanced algorithms to predict future trends and outcomes based on historical data.

Quantum Computing:
Leveraging quantum mechanics principles for faster and more powerful data processing.

Edge Computing:
Processing data closer to its source, reducing latency and improving real-time analytics.

Natural Language Processing (NLP):
Enabling machines to understand and interpret human language for improved communication.

Hybrid Clouds:
Integrating public and private cloud services for flexibility, scalability, and data management.

Dark Data:
Analyzing untapped, unused data to extract valuable insights and enhance decision-making.

Data Fabric:
Creating a unified and connected data environment, allowing seamless data access and integration.

XOps:
Extending DevOps principles to include various aspects of data and technology operations.

3. Real world problem modelling in function style

Functional Style of programming includes various real world applications with clear benefits. Here is a summary of that:

MapReduce Operations:
Problem: Processing large datasets distributed across a cluster using the MapReduce paradigm.

Functional Solution: Model map and reduce operations as pure functions. Emphasize immutability in intermediate data structures. This approach simplifies parallelization and supports fault tolerance.

Data Cleaning and Transformation:
Problem: Cleaning and transforming messy, heterogeneous data sources into a standardized format.

Functional Solution: Develop a series of pure functions for cleaning and transforming individual data elements. Compose these functions in a pipeline, ensuring that each step is modular and independent.

Batch Processing Workflows:
Problem: Designing batch processing workflows for large-scale data analytics.

Functional Solution: Represent each stage of the workflow as a series of functions, emphasizing immutability in data transformations. Use higher-order functions for composing complex workflows. Functional programming makes it easier to reason about the flow of data through the system.

Event Stream Processing:
Problem: Analyzing real-time data streams for timely insights.

Functional Solution: Model stream processing as a series of functions that operate on individual events. Use functional constructs like map, filter, and reduce to process and analyze streaming data. Immutability ensures that each **operation produces a new state.**

Graph Algorithms:
Problem: Analyzing relationships and patterns in large-scale graphs (e.g., social networks, recommendation systems).

Functional Solution: Model graph operations as pure functions, making use of functional constructs like recursion for traversing graphs. Immutability aids in creating algorithms that are easier to reason about and parallelize.

Machine Learning Pipelines:
Problem: Developing machine learning pipelines for training and inference on big datasets

Functional Solution: Represent each step in the machine learning process as a pure function. Compose these functions to create a modular and reusable pipeline. Immutability ensures that models and parameters remain unchanged during processing.

Distributed Caching and State Management:
Problem: Managing distributed state in a scalable and fault-tolerant manner.

Functional Solution: Model state changes as pure functions, making use of immutability to track the evolution of the system. Functional programming aids in handling distributed state across a cluster of nodes.

Concurrency and Parallelism:
Problem: Ensuring efficient parallelism and concurrency in data processing tasks.

Functional Solution: Leverage functional programming features like pure functions and immutability to simplify parallelization. Immutability reduces the need for locks and helps avoid common concurrency issues.

4. With example explain map reduce technique

Map Reduce
It is a framework in which we can write applications to run huge amount of data in parallel and in large cluster of commodity hardware in a reliable manner.

Different Phases of MapReduce:
MapReduce model has three major and one optional phase:
Mapping
Shuffling and Sorting
Reducing
Combining

Mapping: It is the first phase of MapReduce programming. Mapping Phase accepts key-value pairs as input as (k, v), where the key represents the Key address of each record and the value represents the entire record content.The output of the Mapping phase will also be in the key-value format (k’, v’).

Shuffling and Sorting: The output of various mapping parts (k’, v’), then goes into Shuffling and Sorting phase. All the same values are deleted, and different values are grouped together based on same keys. The output of the Shuffling and Sorting phase will be key-value pairs again as key and array of values (k, v[ ]).

Reducer: The output of the Shuffling and Sorting phase (k, v[]) will be the input of the Reducer phase. In this phase reducer function’s logic is executed and all the values are Collected against their corresponding keys. Reducer stabilize outputs of various mappers and computes the final output.

Combining: It is an optional phase in the MapReduce phases . The combiner phase is used to optimize the performance of MapReduce phases. This phase makes the Shuffling and Sorting phase work even quicker by enabling additional performance features in MapReduce phases.

EXAMPLE: Word counting

The text is, *“This is a big data assignment. Big data is very important.”*

The input data is divided into multiple segments, then processed in parallel to reduce processing time. In this case, the input data will be divided into two input splits so that work can be distributed over all the map nodes.
The Mapper counts the number of times each word occurs from input splits in the form of key-value pairs where the key is the word, and the value is the frequency.

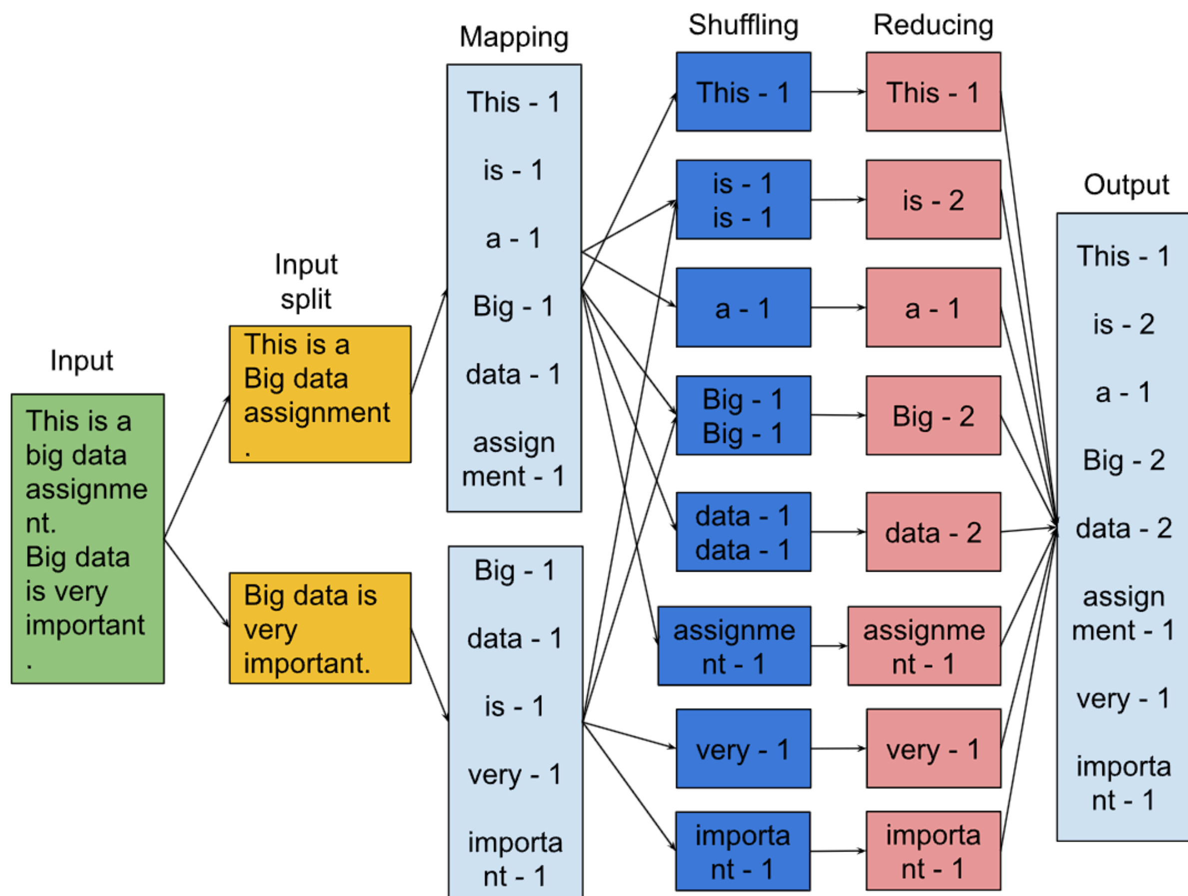


Fig: word counting using MapReduce

For the first input split, it generates 6 key-value pairs: This: 1, is:1, a: 1, Big: 1,data:1, assignment:1. For the second, it generates 5 key-value pairs: Big:1, data:1, is: 1, very:1, important:1.
It is followed by the shuffle phase, in which the values are grouped by keys in the form of key-value pairs. Here we get a total of 8 groups of key-value pairs.

The same reducer is used for all key-value pairs with the same key.
All the words present in the data are combined into a single output in the reducer phase. The output shows the frequency of each word.
Here in the example, we get the final output of key-value pairs as This: 1; is:2 , a:1, Big:2, data:2,assignment:1, very: 1, assignment: 1.
The record writer writes the output key-value pairs from the reducer into the output files, and the final output data is by default stored on HDFS.

5. Explain architecture and working of cassandra, mongodb and HBase

Apache Cassandra
Cassandra is a distributed database management system which is open source with wide column store, NoSQL database to handle large amount of data across many commodity servers which provides high availability with no single point of failure. It is written in Java and developed by Apache Software Foundation. Avinash Lakshman & Prashant Malik initially developed the Cassandra at Facebook to power the Facebook inbox search feature.

Architecture of Apache Cassandra

Basic Terminology:

Node
Node is the basic component in Apache Cassandra. It is the place where actually data is stored. For Example:As shown in diagram node which has IP address 10.0.0.7 contain data (keyspace which contain one or more tables).

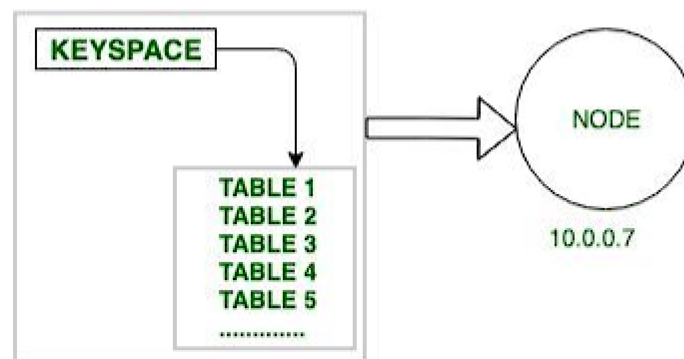


Figure: Node

Data center

Data Center is a collection of nodes.
For example:
DC – N1 + N2 + N3 ….
DC: Data Center
N1: Node 1
N2: Node 2
N3: Node 3

Cluster
It is the collection of many data centers.
For example:
C = DC1 + DC2 + DC3….
C: Cluster
DC1: Data Center 1
DC2: Data Center 2
DC3: Data Center 3

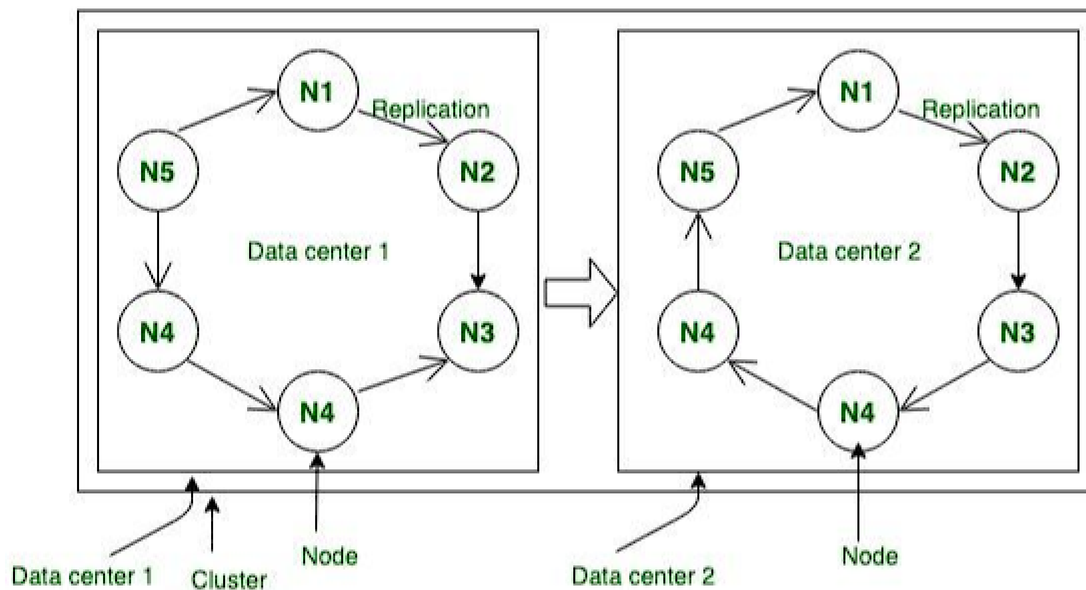


Figure: Node, Data Center, Cluster

Operations:
Read Operation
In Read Operation there are three types of read requests that a coordinator can send to a replica. The node that accepts the write requests called coordinator for that particular operation.

Step-1: Direct Request:
In this operation coordinator node sends the read request to one of the replicas.

Step-2: Digest Request:
In this operation coordinator will contact to replicas specified by the consistency l evel. For Example: CONSISTENCY TWO; It simply means that Any two nodes in data center will acknowledge.

Step-3: Read Repair Request:
If there is any case in which data is not consistent across the node then background Read Repair Request initiated that makes sure that the most recent data is available across the nodes.

Write Operation

Step-1:
In Write Operation as soon as we receives request then it is first dumped into commit log to make sure that data is saved.

Step-2:
Insertion of data into table that is also written in MemTable that holds the data till it's get full.

Step-3:
If MemTable reaches its threshold then data is flushed to SS Table.

Storage Engine:

CommitLog
Commit log is the first entry point while writing to disk or memTable. The purpose of commit log in apache Cassandra is to server sync issues if a data node is down

Memtables
After data written in Commit log then after that data is written in Mem-table. Data is written in Mem-table temporarily.

SSTables
Once Mem-table will reach a certain threshold then data will flushed to the SSTable disk file.

Data Replication Strategies
Basically it is used for backup to ensure no single point of failure. In this strategy Cassandra uses replication to achieve high availability and durability. Each data item is replicated at N hosts, where N is the replication factor configured \per-instance".

There are two type of replication Strategy:
Simple Strategy

In this Strategy it allows a single integer RF (replication_factor) to be defined. It determines the number of nodes that should contain a copy of each row. For example, if replication_factor is 2, then two different nodes should store a copy of each row. It treats all nodes identically, ignoring any configured datacenters or racks.

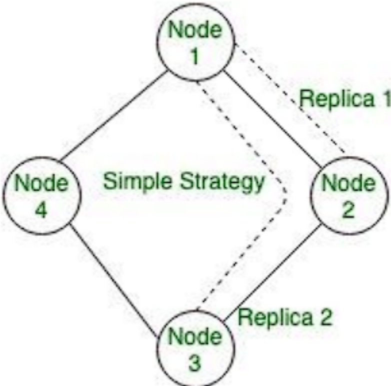


Fig: Simple Strategy

Network Topology Strategy
In this strategy it allows a replication factor to be specified for each datacenter in the cluster. Even if your cluster only uses a single datacenter. This Strategy should be preferred over SimpleStrategy to make it easier to add new physical or virtual datacenters to the cluster later.

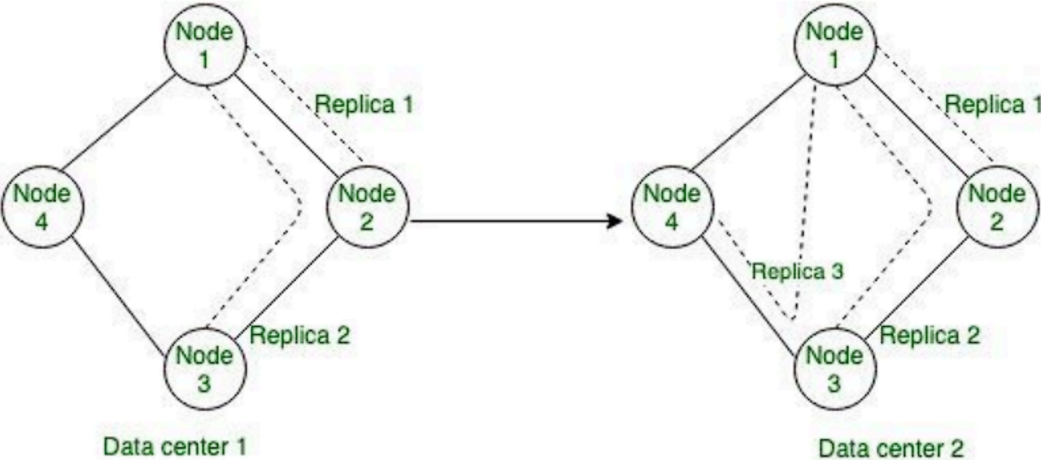


Fig: Network Topology Strategy

Mongodb

MongoDB is a popular NoSQL document-oriented database management system, known for its flexibility, high performance, high availability, and multi-storage engines. The term NoSQL means non-relational. It means that MongoDB isn't based on a table-like relational database structure. It is used by Adobe, Uber, IBM, and Google. In this article, we will delve into the MongoDB architecture, exploring its key components and how they work together.

Architecture of Mongodb
MongoDB’s architecture design involves several important parts that work together to create a strong and flexible database system. these are the following MongoDB’s architecture.

Drivers & Storage Engine
MongoDB store the data on the server but that data we will try to retrieve from our application. So that time how the communication is happening between our application and MongoDB server.
Any application which is written in python, .net and java or any kind of frontend application, these application are trying to access the data from these physical storage in server. First they will interact with driver which will communicate with MongoDB server. What happen is once the request is going from the frontend application through the driver then driver will change appropriate query by using query engine and then the query will get executed in MongoDB data model. Left side is security which provides security to the database that who will access the data and right side is management this management will manage all these things.

Drivers
Drivers are client libraries that offer interfaces and methods for applications to communicate with MongoDB databases. Drivers will handle the translation of documents between BSON objects and mapping application structures..NET, Java, JavaScript, Node.js, Python, etc are some of the widely used drives supported by MongoDB.

Storage Engine
The storage engine significantly influences the performance of applications, serving as an intermediary between the MongoDB database and persistent storage, typically disks. MongoDB supports different storage engines:

MMAPv1 – It is a traditional storage engine based on memory mapped files. This storage engine is optimized for workloads with high volumes of read operations, insertions, and in-place updates. It uses B-tress to store indexes. Storage Engine works on multiple reader single writer lock. A user cannot have two write calls to be processes in parallel on the same collection. It is fast for reads and slow for writes.

Wired Tiger – Default Storage Engine starts from MongoDB 3version. No locking Algorithms like hash pointer. It yields 7x-10x better write operations and 80% of the file system compression than MMAP.

InMemory – Instead of storing documents on disk, the engine uses in-memory for more predictable data latencies. It uses 50% of physical

RAM minimum 1 GB as default. It requires all its data. When dealing with large datasets, the in-memory engine may not be the most suitable choice.

Security

1. Authentication
2. Authorization
3. Encryption on data
4. Hardening (Ensure only trusted hosts have access)

Mongodb server

It serves as the central element and is in charge of maintaining, storing, and retrieving data from the database through a number of interfaces. The system's heart is the MongoDB server. Each mongod server instance is in charge of handling client requests, maintaining data storage, and performing database operations. Several mongod instances work together to form a cluster in a typical MongoDB setup.
MongoDB Shell
For dealing with MongoDB databases, MongoDB provides the MongoDB Shell command-line interface (CLI) tool. The ability to handle and query MongoDB data straight from the terminal is robust and flexible. After installing MongoDB, you may access the MongoDB Shell, often known as mongo. It interacts with the database using JavaScript-based syntax. Additionally, it has built-in help that shows details about possible commands and how to use them.

Data Storage in MongoDB

Collections

A database can contain as many collections as it wishes, and MongoDB stores data inside collections.
As an example, a database might contain three collections a user's collection, a blog post collection, and a comments collection. The user collection would hold user data and documents, the blog post collection would hold blog posts and documents, and the comments collection would hold documents related to comments. This would allow for the easy retrieval of all the documents from a single collection.

Documents

Documents themselves represent the individual records in a specific collection. For example inside the blog posts collection we'd store a lot of blog post documents and each one represents a single blog post now the way that data is structured inside a document looks very much like a JSON object with key value pairs but actually it's being stored as something called BSON which is just binary JSON.

Indexes

Indexes are data structures that make it simple to navigate across the collection's data set. They help to execute queries and find documents that match the query criteria without a collection scan.

These are the following different types of indexes in MongoDB:

- **Single field**
  MongoDB can traverse the indexes either in the ascending or descending order for single-field index
  db.students.createIndex({"item":1})
  In this example, we are creating a single index on the item field and 1 here represents the filed is in ascending order.
  A compound index in MongoDB contains multiple single filed indexes separated by a comma. MongoDB restricts the number of fields in a compound index to a maximum of 31.
  db.students.createIndex({"item": 1, "stock":1})
  Here, we create a compound index on item: 1, stock:1

- **Multi-Key**
  When indexing a filed containing an array value, MongoDB creates separate index entries for each array component. MongoDB allows you to create multi-key indexes for arrays containing scalar values, including strings, numbers, and nested documents.
  db.students.createIndex({<filed>: <1 or -1>})

- **Geo Spatial**
  Two geospatial indexes offered by MongoDB are called 2d indexes and 2d sphere indexes. These indexes allow us to query geospatial data. On this case, queries intended to locate data stored on a two-dimensional plane are supported by the 2d indexes. On the other hand, queries that are used to locate data stored in spherical geometry are supported by 2D sphere indexes.

- **Hashed**
  To maintain the entries with hashes of the values of the indexed field we use Hash Index. MongoDB supports hash based sharding and provides hashed indexes.
  db.<collection>.createIndex({ item: "hashed" })

- **Replication**
  Within a MongoDB cluster, data replication entails keeping several copies of the same data on various servers or nodes. Enhancing data availability and dependability is the main objective of data replication. A replica may seamlessly replace a failing server in the cluster to maintain service continuity and data integrity.

- **Primary Node (Primary Replica):** In a replica set, the primary node serves as the main source for all write operations. It's the only node that accepts write requests. The main node is where all data modifications begin and are implemented initially.

- **Secondary Nodes:** Secondary nodes duplicate data from the primary node (also known as secondary replicas). They are useful for dispersing read workloads and load balancing since they are read-only and mostly utilized for read activities.

Sharding

Sharding is basically horizontal scaling of databases as compared to the traditional vertical scaling of adding more CPUs and RAM to the current system.

For example, you have huge set of files you might segregate it into smaller sets for ease. Similarly what mongo database does is it segregates its data into smaller chunks to improve the efficiency. You have a machine with these configuration and mongo db instance running on it storing 100 million documents.

Now with time your data will grow in your mongo db instance and suppose 100 million extra documents get added. Now to manage the processing of these extra records you might need to add extra ram, extra storage and extra CPU to the server. Such type of scaling is called vertical scaling.

Now consider another situation if you have 4 small machines with small configurations. You can divide 200 million of document into each of the server such that each of the server might hold around 50 million documents. By dividing the data into multiple servers you have reduced the computation requirements and such kind of scaling is known as horizontal scaling and this horizontal scaling is known as sharding in mongo and each of the servers S1, S2, S3, S4 are the shards.

The partioning of data in a sharded environment is done on a range based basis by deciding a field as a shard key.

Hbase

HBase is a data model that is similar to Google’s big table. It is an open source, distributed database developed by Apache software foundation written in Java. HBase is an essential part of our Hadoop ecosystem. HBase runs on top of HDFS (Hadoop Distributed File System). It can store massive amounts of data from terabytes to petabytes. It is column oriented and horizontally scalable.

Architecture of Hbase

HBase architecture has 3 main components: HMaster, Region Server, Zookeeper.

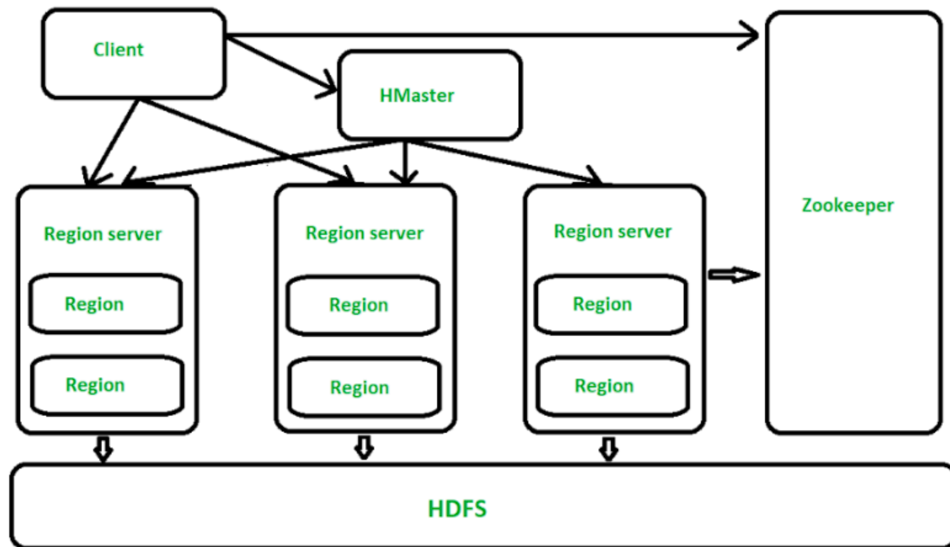


Fig: Architecture of HBase

HMaster:
The implementation of Master Server in HBase is HMaster. It is a process in which regions are assigned to region server as well as DDL (create, delete table) operations. It monitor all Region Server instances present in the cluster. In a distributed environment, Master runs several background threads. HMaster has many features like controlling load balancing, failover etc.

Region Server:
HBase Tables are divided horizontally by row key range into Regions. Regions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. Region Server runs on HDFS DataNode which is present in Hadoop cluster. Regions of Region Server are responsible for several things, like handling, managing, executing as well as reads and writes HBase operations on that set of regions. The default size of a region is 256 MB.

Zookeeper:
It is like a coordinator in HBase. It provides services like maintaining configuration information, naming, providing distributed synchronization, server failure notification etc. Clients communicate with region servers via zookeeper.

6. With example explain distributed searching with elastic search

Logical Concepts

- **Documents**
  Documents in Elasticsearch serve as fundamental units of information, stored in JSON format for global data interchange. Similar to rows in a relational database, each document represents a specific entity, such as an encyclopedia article or web server log entries. These documents encompass more than just text, accommodating structured data like numbers, strings, and dates. Each document is uniquely identified and associated with a particular data type, defining the nature of the represented entity.

- **Indices**
  An index in Elasticsearch is a grouping of documents sharing similar characteristics. It serves as the top-level entity for queries, analogous to a database in a relational schema. Documents within an index are logically related; for instance, an e-commerce website might have separate indexes for Customers, Products, and Orders. Each index is identified by a name, used for operations like indexing, searching, updating, and deleting documents associated with it.

- **Inverted Index**
  In Elasticsearch, an index operates as an inverted index, a core concept utilized by search engines. It serves as a data structure that establishes a mapping from content, be it words or numerical data, to their specific locations within one or more documents. Rather than directly storing strings, the inverted index dissects each document into individual search terms, associating each term with the documents in which it is present. For example, if the term "mountain" appears in document 3, it is mapped accordingly. This approach enables efficient search term retrieval within documents, allowing Elasticsearch to swiftly pinpoint optimal matches in vast datasets through distributed inverted indices.

Backend Components

- **Cluster**
  An Elasticsearch cluster refers to a collective assembly of one or more interconnected node instances. The strength of an Elasticsearch cluster stems from the collaborative distribution of tasks, including searching and indexing, across all nodes within the cluster. This distributed architecture enhances efficiency and scalability in handling diverse operations.

- **Node**
  A node is a single server that is a part of a cluster. A node stores data and participates in the cluster’s indexing and search capabilities. An Elasticsearch node can be configured in different ways:

  - **Master Node** — Controls the Elasticsearch cluster and is responsible for all cluster-wide operations like creating/deleting an index and adding/removing nodes.

  - **Data Node** — Stores data and executes data-related operations such as search and aggregation.

  - **Client Node** — Forwards cluster requests to the master node and data-related requests to data nodes.

- **Shards**
  Elasticsearch offers the feature of dividing an index into multiple segments known as shards. Each shard acts as a standalone and fully operational "index," capable of residing on any node within a cluster. Through the distribution of index documents across multiple shards and dispersing those shards across various nodes, Elasticsearch achieves redundancy. This not only safeguards against hardware failures but also enhances query capacity by accommodating additional nodes in a cluster.

- **Replicas**
  Elasticsearch enables the creation of replicas for index shards, known as "replica shards" or simply "replicas." Essentially, a replica shard functions as a duplicate of a primary shard. Every document within an index belongs to a primary shard. Replicas serve the purpose of offering redundant copies of data, serving as a safeguard against hardware failures, and enhancing the capacity to fulfill read requests, such as searching or retrieving documents.

7. Explain Hadoop

Installation on different environment

I followed the website below to install hadoop in wsl on windows in pseudo-distributed mode:

https://kontext.tech/article/978/install-hadoop-332-in-wsl-on-windows

Architecture

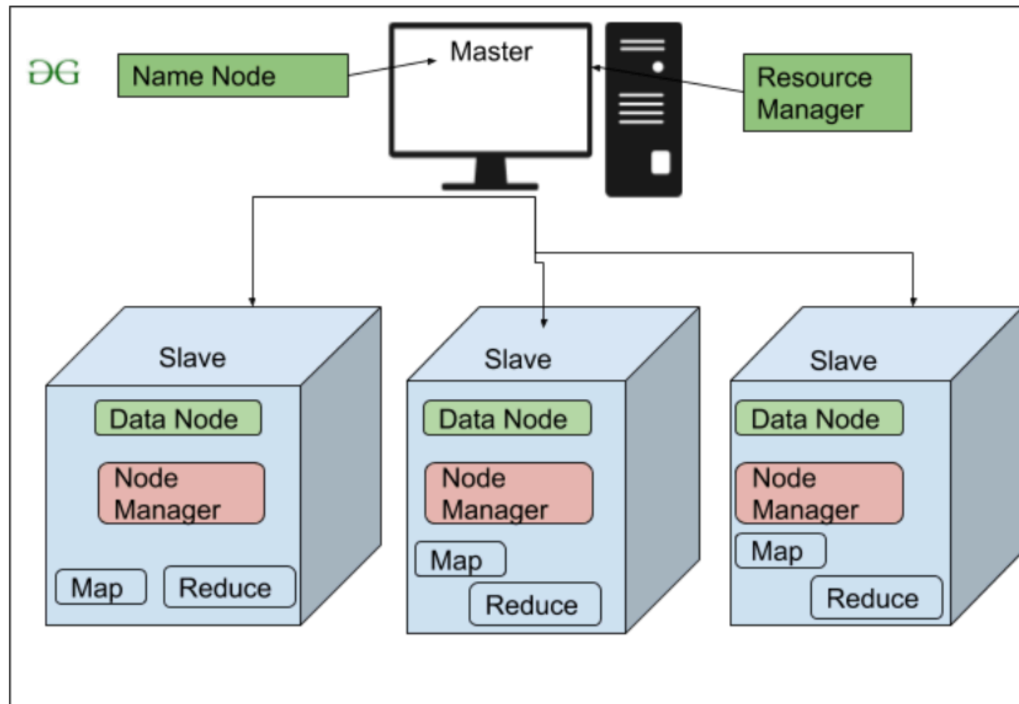


Fig: Architecture of Hadoop

The Hadoop Architecture Mainly consists of 4 components.

- MapReduce
- HDFS(Hadoop Distributed File System)
- YARN(Yet Another Resource Negotiator)
- Common Utilities or Hadoop Common

Mapreduce

It is a framework in which we can write applications to run huge amount of data in parallel and in large cluster of commodity hardware in a reliable manner

HDFS(Hadoop Distributed File System)

HDFS(Hadoop Distributed File System) is utilized for storage permission. It is mainly designed for working on commodity Hardware devices(inexpensive devices), working on a distributed file system design. HDFS is designed in such a way that it believes more in storing the data in a large chunk of blocks rather than storing small data blocks.

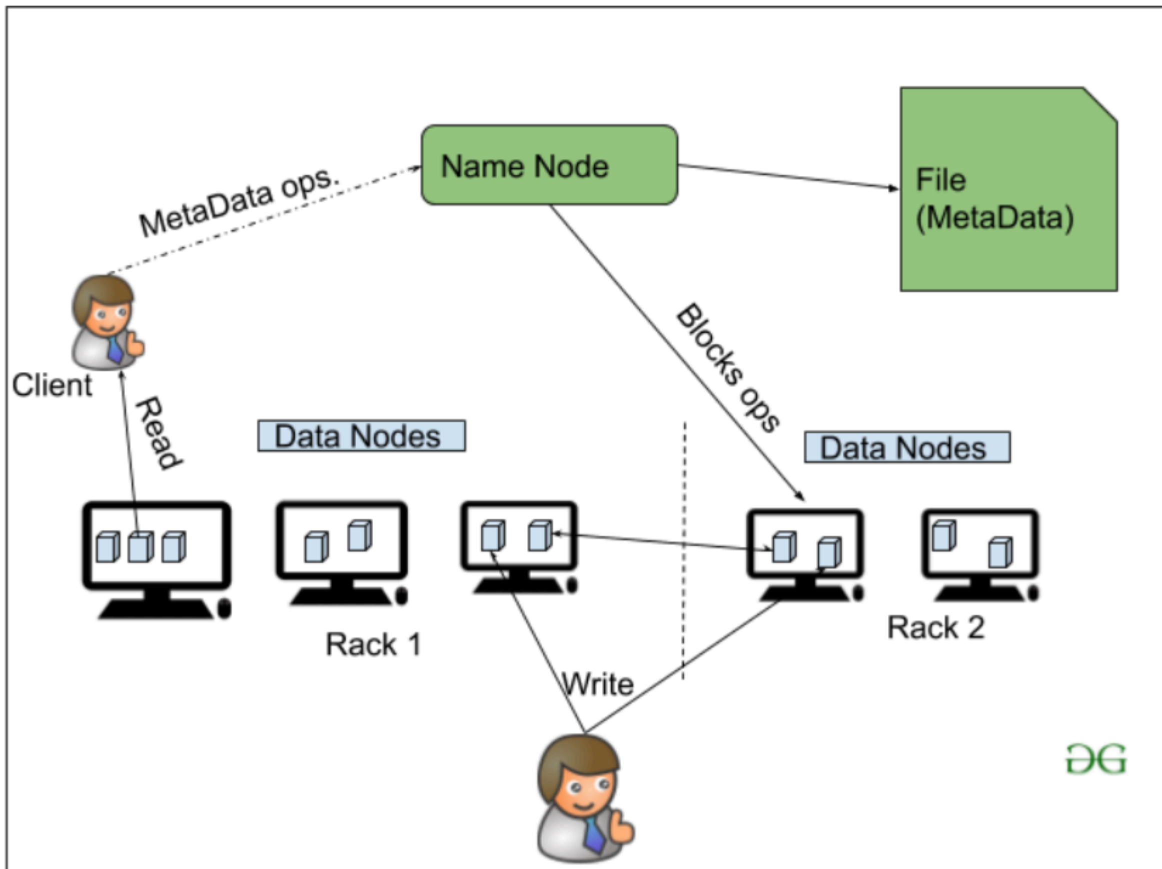


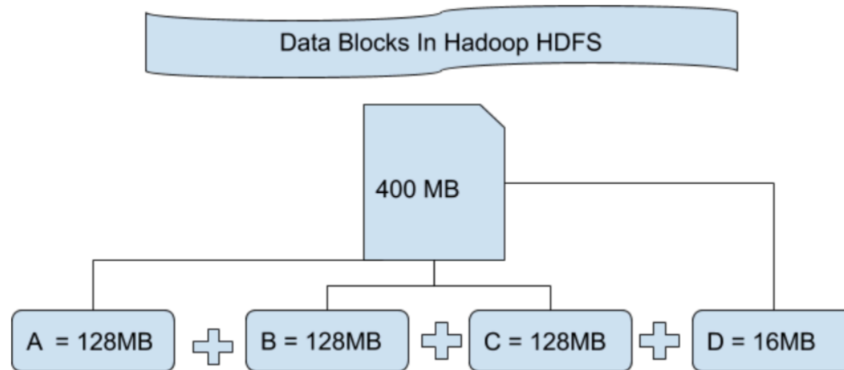
Fig: Architecture of HDFS

HDFS in Hadoop provides Fault-tolerance and High availability to the storage layer and the other devices present in that Hadoop cluster. Data storage Nodes in HDFS.

- **NameNode:** NameNode works as a Master in a Hadoop cluster that guides the Datanode(Slaves). Namenode is mainly used for storing the Metadata i.e. the data about the data. Meta Data can be the transaction logs that keep track of the user's activity in a Hadoop cluster.
  Meta Data can also be the name of the file, size, and the information about the location(Block number, Block ids) of Datanode that Namenode stores to find the closest DataNode for Faster Communication. Namenode instructs the DataNodes with the operation like delete, create, Replicate, etc.

- **DataNode:** DataNodes works as a Slave DataNodes are mainly utilized for storing the data in a Hadoop cluster, the number of DataNodes can be from 1 to 500 or even more than that. The more number of DataNode, the Hadoop cluster will be able to store more data. So it is advised that the DataNode should have High storing capacity to store a large number of file blocks.

File Block In HDFS: Data in HDFS is always stored in terms of blocks. So the single block of data is divided into multiple blocks of size 128MB which is default and you can also change it manually.



blocks of size 128MB which is default and you can also change it manually.

Replication: Replication ensures the availability of the data. Replication is making a copy of something and the number of times you make a copy of that particular thing can be expressed as it's Replication Factor. As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks.

By default, the Replication Factor for Hadoop is set to 3 which can be configured means you can change it manually as per your requirement like in above example we have made 4 file blocks which means that 3 Replica or copy of each file block is made means total of 4×3 = 12 blocks are made for the backup purpose.

This is because for running Hadoop we are using commodity hardware (inexpensive system hardware) which can be crashed at any time. We are not using the supercomputer for our Hadoop setup. That is why we need such a feature in HDFS which can make copies of that file blocks for backup purposes, this is known as fault tolerance.
Now one thing we also need to notice that after making so many replica's of our file blocks we are wasting so much of our storage but for the big brand organization the data is very much important than the storage so nobody cares for this extra storage. You can configure the Replication factor in your hdfs-site.xml file.

Rack Awareness: The rack is nothing but just the physical collection of nodes in our Hadoop cluster (maybe 30 to 40). A large Hadoop cluster is consists of so many Racks . with the help of this Racks information Namenode chooses the

closest Datanode to achieve the maximum performance while performing the read/write information which reduces the Network Traffic.

YARN(Yet Another Resource Negotiator)

YARN is a Framework on which MapReduce works. YARN performs 2 operations that are Job scheduling and Resource Management. The Purpose of Job schedular is to divide a big task into small jobs so that each job can be assigned to various slaves in a Hadoop cluster and Processing can be Maximized. Job Scheduler also keeps track of which job is important, which job has more priority, dependencies between the jobs and all the other information like job timing, etc. And the use of Resource Manager is to manage all the resources that are made available for running a Hadoop cluster.

Features of YARN

- Multi-Tenancy
- Scalability
- Cluster-Utilization
- Compatibility

Common Utilities or Hadoop Common

Hadoop common or Common utilities are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster. these utilities are used by HDFS, YARN, and MapReduce for running the cluster. Hadoop Common verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

Working of HDFS

- HDFS divides the client input data into blocks of size 128 MB. Depending on the replication factor, replicas of blocks are created. The blocks and their replicas are stored on different DataNodes
- Once all blocks are stored on HDFS DataNodes, the user can process the data.
- To process the data, the client submits the MapReduce program to Hadoop.
- ResourceManager then scheduled the program submitted by the user on individual nodes in the cluster.
- Once all nodes complete processing, the output is written back to the HDFS.

8. Map reduce with hadoop with example and code

I have following text as input.txt

```
Howdy Fellers!
Sheriff Prashant here is the best Sheriff round town. The town ain’t
big enough for two Sheriff Prashnt. It sure is great when he says
Howdy Fellers! Yall gone done dadgum done did it now Fellers?
But everything changed when the Fire Fellers attacked
```

WordCount.java

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
```

```
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Steps to run map reduce WordCount.java program on hadoop:

Step 1: Create an Input Directory in hdfs
hdfs dfs -mkdir /Input

Step 2: Create input.txt file in your current directory with some contents in it.

Step 3: Put input.txt from your current directory into /Input of hdfs
hdfs dfs -put input.txt /Input

Step 4: Compile WordCount.java assuming WordCount.java is in current directory
javac -source 1.8 -target 1.8 -classpath $(hadoop classpath) -d WordCount/ WordCount.java

After compilation, the class files will be generated inside WordCount folder, which is created as the subfolder of current directory.

Step 5: Generate a jar file WordCount.jar
jar cvf WordCount.jar -C WordCount/ .

Step 6: Run the WordCount.jar file using hadoop to generate the output
hadoop jar WordCount.jar WordCount /Input /Output

Step 7: Display the output in the terminal
hdfs dfs -cat /Output/part-r-00000

When we use command " hdfs dfs -ls / ", we can see input,output and temp folder. Input folder contains input.txt where as Ouput has complied output of wordcount.java.

```
fluxo4@Fluxo4:~/hadoop/hadoop-3.3.6$ hdfs dfs -ls /
Found 3 items
drwxr-xr-x   - fluxo4 supergroup          0 2024-02-20 18:10 /Input
drwxr-xr-x   - fluxo4 supergroup          0 2024-02-20 18:12 /Output
drwxr-xr-x   - fluxo4 supergroup          0 2024-02-19 20:35 /tmp
```

```
fluxo4@Fluxo4:~/hadoop/hadoop-3.3.6$ hdfs dfs -ls /Output
Found 2 items
-rw-r--r--   1 fluxo4 supergroup          0 2024-02-20 18:12 /Output/_SUCCESS
-rw-r--r--   1 fluxo4 supergroup        250 2024-02-20 18:12 /Output/part-r-00000
fluxo4@Fluxo4:~/hadoop/hadoop-3.3.6$ hdfs dfs -ls /Input
Found 1 items
-rw-r--r--   1 fluxo4 supergroup        261 2024-02-20 18:10 /Input/input.txt
```

Final Output:

```
fluxo4@Fluxo4:~/hadoop/hadoop-3.3.6$ hdfs dfs -cat /Output/part-r-00000
howdy:      2
fellers!:   2
sheriff:    3
prashant:   1
here: 1
is:   2
the:  3
best: 1
round:      1
town: 2
ain't:      1
big:  1
enough:     1
for:  1
two:  1
prashant.:  1
it:   2
sure: 1
great:      1
when: 2
he:   1
says: 1
yall: 1
gone: 1
done: 2
dadgum:     1
did:  1
now:  1
fellers?:   1
but:  1
everything: 1
changed:    1
fire: 1
fellers:    1
attacked:   1
```