UNIT II

Chapter 3

Introduction to NOSQL

1. **Introduction**:
Non-relational databases ("NoSQL databases") are becoming popular with the increasing use of cloud computing services. Non-relational databases have better horizontal scaling capability and improved performance for big data at the cost of having less rigorous consistency models. NoSQL databases are popular for applications in which the scale of data involved is massive and the data may not be structured. Furthermore, real-time performance is more important than consistency. These systems are optimized for fast retrieval and appending operations on records. Unlike relational databases, the NoSQL databases do not have a strict schema. The records can be in the form of key-value pairs or documents. Most NoSQL databases are classified in terms of the data storage model or type of records that can be stored
NoSQL refers to all databases and data stores that are not based on the Relational Database Management Systems or RDBMS principles. It relates to large data sets accessed and manipulated on a Web scale. NoSQL does not represent single product or technology. It represents a group of products and a various related data concepts for storage and management. With the explosion of social media, user-driven content has grown rapidly and has increased the volume and type of data that is produced, managed, analyzed, and archived. In addition, new sources of data, such as sensors, Global Positioning Systems or GPS, automated trackers, and other monitoring systems generate huge volumes of data on a regular basis.
These large volumes of data sets, also called big data, have introduced new challenges and opportunities for data storage, management, analysis, and archival. In addition, data is becoming increasingly semi-structured and sparse. This means that RDBMS databases which require upfront schema definition and relational references are examined.

RDBMS	NoSQL
• Data is stored in a relational model, with rows and columns. • A row contains information about an item while columns contain specific information, such as 'Model', 'Date of Manufacture', 'Color'. • Follows fixed schema. Meaning, the columns are defined and locked before data entry. In addition, each row contains data for each column. • Supports vertical scaling. Scaling an RDBMS across multiple servers is a challenging and time-consuming process. • Atomicity, Consistency, Isolation &	• Data is stored in a host of different databases, with different data storage models. • Follows dynamic schemas. Meaning, you can add columns anytime. • Supports horizontal scaling. You can scale across multiple servers. Multiple servers are cheap commodity hardware or cloud instances, which make scaling cost-effective compared to vertical scaling. • Not ACID Compliant.

Durability(ACID) Compliant	

2. **History of NOSQL:**
   The NOSQL database was invented by Carlo Strozzi in 1998. It provides a mechanism for retrieval and storage of data. Most of the NOSQL databases are non-relational, distributed and do not follow ACID properties. Examples of NoSQL databases are MongoDB, Apache Cassandra and Neo4j

3. **Characteristics of NOSQL:**
   NOSQL databases do not use SQL. The creators of NOSQL meant that it is a query language that doesn't follow the principles of RDBMS. They are open-source projects. NoSQL databases are driven by the need to run on clusters. NOSQL databases offer a range of options for consistency and distribution. We can freely add fields to the database records without having to define any change to the structure because NoSQL operate without a schema. This is useful while dealing with non-uniform data.
4. **Advantages of NOSQL:**
   The following are the advantages of NOSQL
   a. Can easily scale up and down: NoSQL database supports scaling rapidly and elastically. It also allows to the cloud.
      - Cluster scale: It allows distribution of database across 100+ nodes often in multiple data centres
      - Performance scale: It sustains over 100,000+ database reads and writes per second.
      - Data Scale: It supports housing of 1 billion+ documents in the database.

   b. Doesn't require pre-defined schema: NoSQL does not require any adherence to pre-defined schema. It is very flexible.
   c. Cheap and easy to implement: Deploying NoSQL allows for benefits if scale, high availability and fault tolerance thereby lowering the operational costs

d. Relaxes data consistency requirement: NoSQL databases adhere to CAP theorem. Most NoSQL databases compromise in consistency in favour of availability and partition tolerance.

e. Data can be replicated to multiple nodes and can be partitioned: This advantage can be explained using the following terms:
   - Sharding: Here different pieces if data are distributed across multiple servers. NoSQL databases support auto sharding i.e. they can natively and automatically spread across an arbitrary number of servers, without requiring the application to even be aware of the composition of the server pool. Servers can be added or removed from the data layer without application downtime. This would mean that data and query load are automatically balanced across servers and when servers go down, it can be quickly and transparently replaced with no application disruption
   - Replication: Multiple copies of data are stored across clusters and even across data centers. This promises high availability and fault tolerance.

5. **Applications of NOSQL:**

NOSQL is being put to use for varied reasons in industries. They are used to support analysis for applications such as web user data analysis, log analysis , sensor feed analysis, making recommendations for upsell and cross sell.

Types of NoSQL database	Application
Key-Value Pair databases	Shopping carts, web user data
Column Oriented Databases	Analyze huge web user actions, sensor feeds
Document based Databases	Real time analytics, logging, document archive management
Graph databases	Network modelling and recommendations for companies

Popular NoSQL vendors

Company	**Product**	**Widely used**
Amazon	Dynamo DB	LinkedIn, Mozilla
Facebook	Cassandra	Netflix, Twitter, ebay
Google	BigTable	Adobe Photoshop

6. Classification of NOSQL:

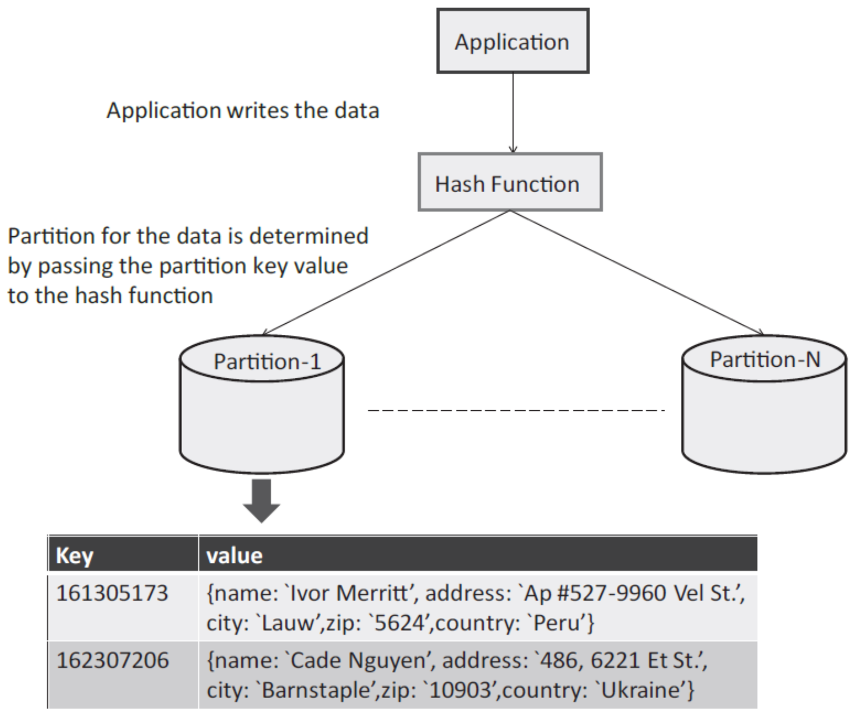
6.1 Key-Value Databases:

Key-value databases are the simplest form of NoSQL databases. These databases store data in the form of key-value pairs. The keys are used to identify uniquely the values stored in the database. Applications that want to store data generate unique keys and submit the key-value pairs to the database. The database uses the key to determine where the value should be stored. Most key-value databases have distributed architectures comprising of multiple storage nodes. The data is partitioned across the storage nodes by the keys. For determining the partitions for the keys, hash functions are used. The partition number for a key is obtained by applying a hash function to the key. The hash functions are chosen such that the keys are evenly distributed across the partitions.

Key-value databases provide a lot of flexibility in terms of the type of values that can be stored. The values can be virtually of any type (such as strings, integers, floats, binary large object (BLOB), etc.). Most key-value stores have support for native programming language data types. There are limits on the size of the values that can be stored. Unlike relational databases in which the tables have fixed schemas and there are constraints on the columns, in key-value databases, there are no such constraints. Key-value databases do not have tables like in relational databases. However, some key-value databases support tables, buckets or collections to create separate namespaces for the keys. Keys within a table, bucket or collection are unique. Key-value databases are suited for applications that require storing unstructured data without a fixed schema. These databases can be scaled up horizontally and can store a very large number of key-value pairs. Unlike relational databases which provide specialized query languages (such as SQL), the key-value databases only provide basic querying and searching capabilities. Key-value databases are suitable for applications for which the ability to store and retrieve data in a fast and efficient manner is more important than imposing structure or onstraints on the data. For example, key-value databases can be used to store configuration data, user data, transient or intermediate data (such as shopping cart data), item- attributes and BLOBs (such as audio and images).

6.1.1. **Amazon DynamoDB**

Amazon DynamoDB is a fully-managed, scalable, high-performance NoSQL database service from Amazon. DynamoDB provides fast and predictable performance and seamless scalability without any operational overhead. DynamoDB is an excellent choice for a serving database for data analytics applications as it allows storing and retrieving any amount of data and the ability to scale up or down the provisioned throughput depending on the application’s performance requirements. DynamoDB is a highly available and reliable service. The data stored in DynamoDB is replicated across multiple availability zones.


Application
Application writes the data
Hash Function
Partition for the data is determined by passing the partition key value to the hash function
Partition-1
Partition-N

Key	value
161305173	{name: `Ivor Merritt’, address: `Ap #527-9960 Vel St.’, city: `Lauw’,zip: `5624’,country: `Peru’}
162307206	{name: `Cade Nguyen’, address: `486, 6221 Et St.’, city: `Barnstaple’,zip: `10903’,country: `Ukraine’}

DynamoDB’s data model includes Tables, Items, and Attributes. A table is a collection of items and each item is a collection of attributes. Tables in DynamoDB do not have a fixed schema. While creating a table, only the primary key needs to be specified. The primary key uniquely identifies the items in a table. The primary key is a combination of a partition key and an optional sort key. The partition key is hashed using a hash function to determine the partition where the item should be stored. The partition key value must be unique across all items if no sort is specified. An optional sort key can be specified which is used to sort items within a partition. If the primary key used is a combination of the hash key and sort key then it is possible for two items to have the same value of the partition key, but the sort key must have different values. Items are composed of attributes. The attributes can be added at runtime. Different items in a table can have different attributes. Each attribute is a key-value pair.

For reading items, DynamoDB provides scan and query operations. The scan operation is used to retrieve all items in the table. You can specify optional filtering criteria. The filtering criteria can look for specific values of attributes or range of values. The query operation is used to query for items with the primary key (either only the partition key or the partition key and the sort key). To query the table using attributes other than the primary key, secondary indexes can be added.

6.2 Document Databases:

Document store databases store semi-structured data in the form of documents which are encoded in different standards such as JSON, XML, BSON or YAML. By semi-structured data we mean that the documents stored are similar to each other (similar fields, keys or attributes) but there are no strict requirements for a schema. Documents are organized in different ways in different document database such in the form of collections, buckets or tags.

Each document stored in a document database has a collection of named fields and their values. Each document is identified by a unique key or ID. There is no need to define any schema for the documents before storing them in the database. While it is possible to store JSON or XML-like documents as values in a key-value database, the benefit of using document databases over key-value databases is that these databases allow efficiently querying the documents based on the attribute values in the documents. Document databases are useful for applications that want to store semi-structured data with a varying number of fields.

6.2.1 MongoDB

MongoDB is a document-oriented non-relational database system. MongoDB is powerful, flexible and highly scalable database designed for web applications and is a good choice for a serving database for data analytics applications. The basic unit of data stored by MongoDB is a document. A document includes a JSON-like set of key-value pairs.

Documents are grouped together to form collections. Collections do not have a fixed schema and different documents in a collection can have different sets of key-value pairs. Collections are organized into databases, and there can be multiple databases running on a single MongoDB instance.

6.3 Column Family Databases

In column family databases the basic unit of data storage is a column, which has a name and a value. A collection of columns make up a row which is identified by a row-key. Columns are grouped together into columns families. Unlike, relational databases, the column family databases do not need to have fixed schemas and a fixed number of columns in each row. The number of columns in a column family database can vary across different rows. A column family can be considered as a map having key-value pairs and this map can vary across different rows. Column family databases store data in a denormalized form so that all relevant information related to an entity required by the applications can be retrieved by reading a single row. Column family databases support high-throughput reads and writes and have distributed and highly available architectures.

6.3.1 HBase

HBase is a scalable, non-relational, distributed, column-family database that provides structured data storage for large tables. HBase can store both structured and unstructured data. The data storage in HBase can scale linearly and automatically by the addition of new nodes. HBase has been designed to work with commodity hardware and is a highly reliable and fault tolerant system. HBase allows fast random reads and writes.

6.3.2 HBase Model:

The figure shows the structure of an HBase table. A table is consists of rows, which are indexed by the row key. Each row includes multiple column families. Each column family includes multiple columns. Each column includes multiple cells or entries which are timestamped. HBase tables are indexed by the row key, column key and timestamp. Unlike relational database tables, HBase tables do not have a fixed schema. HBase columns families are declared at the time of creation of the table and cannot be changed later. Columns can be added dynamically, and HBase can have millions of columns.

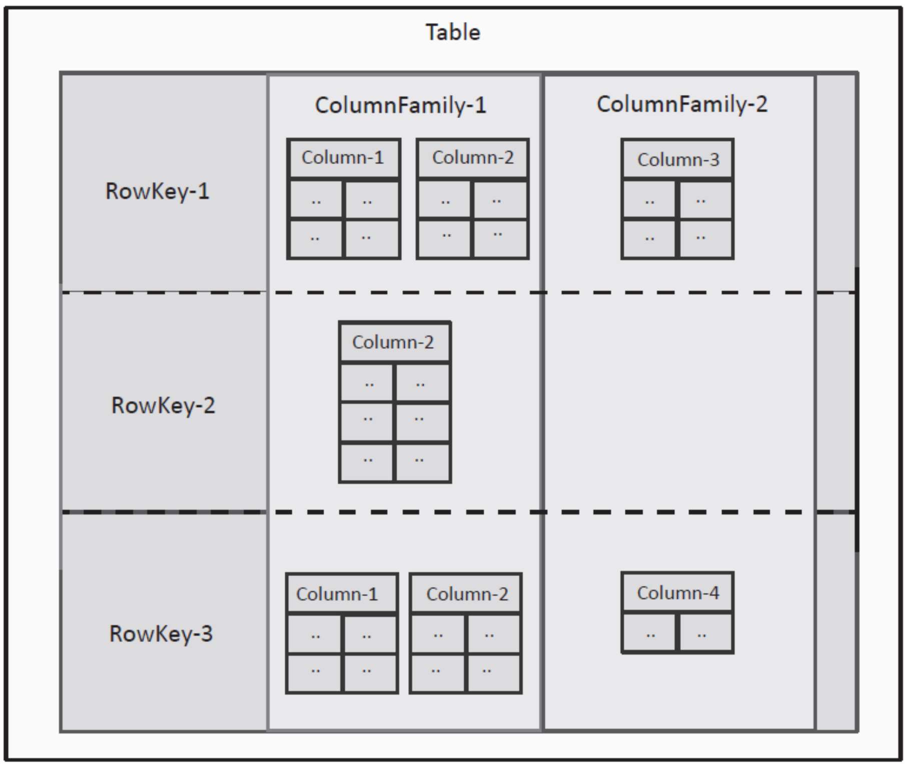


Figure 4.5: HBase table structure

Key Length	Value Length	Row Length	Row Key	Column Family Length	Column Family	Column Qualifier	Time Stamp	Key Type	Value

Figure 4.6: HBase key-value format

HBase is often described as a sparse, distributed, persistent, multi-dimensional and sorted map. The features of HBase are:

- Sparse: In traditional relational databases, tables have fixed schemas. Each row in a table has the same number of columns. Each row has all the columns even if all of them are not populated. HBase, in contrast, has sparse tables as each row doesn't need to have all the columns. Only the columns which are populated in a row are stored.
- Distributed: HBase is a distributed database. HBase tables are partitioned based on row keys into regions. Each region contains a range of row keys. A typical HBase deployment contains multiple Region Servers. Each Region Server contains several regions from different tables.
- Persistent: HBase works on top of HDFS and all data stored in HBase tables is persisted on HDFS.
- Multi-dimensional : HBase stores data as key-value pairs where the keys are multi-dimensional. A key includes: (Table, RowKey, ColumnFamily, Column, TimeStamp) as shown in Figure 4.6. For each entry/cell, multiple versions are stored, which are timestamped.
- Sorted Map: HBase rows are sorted by the row key in lexicographic order. Columns in a column family are sorted by the column key.

HBase cells cannot be over-written. Since the cells are versioned with timestamps, when newer values are added, the older values are also retained. Data is stored in cells as byte arrays. The applications are responsible for correctly interpreting the data type.

HBase Architecture

HBase has a distributed architecture as shown in Figure. An HBase deployment comprises multiple region servers which usually run on the same machines as the Hadoop data nodes. HBase tables are partitioned by the row key into multiple regions (HRegions). Each region server has multiple regions. HBase has a master-slave architecture with one of the nodes acting as the master node (HMaster) and other nodes are slave nodes. The HMaster is responsible for maintaining the HBase meta-data and assignment of regions to region servers. HBase uses Zookeeper for distributed state coordination. HBase has two special tables - ROOT and META, for identifying which region server is responsible for serving a read/write request for a specific row key.

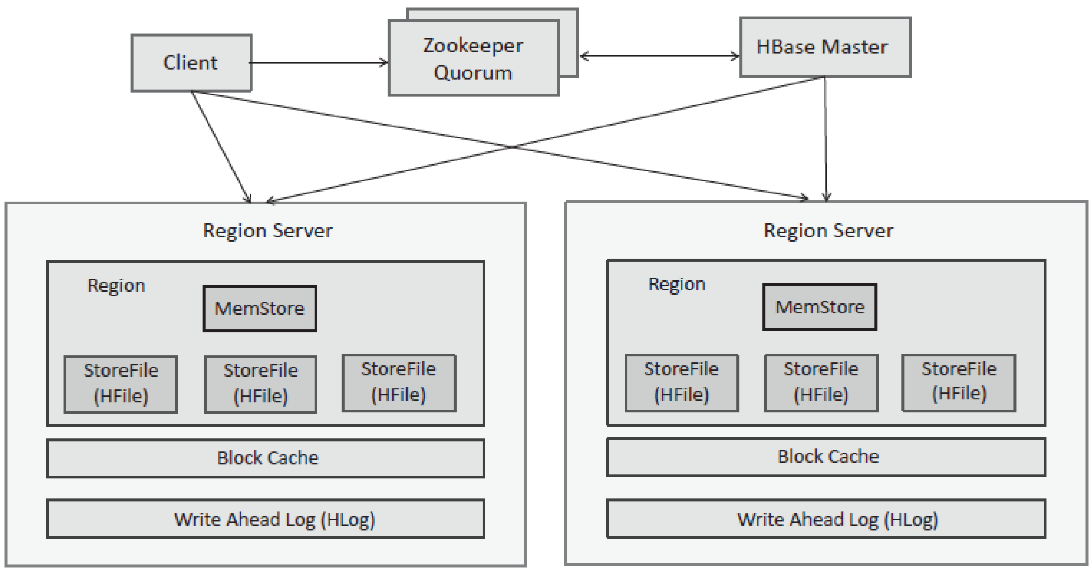


Figure 4.7: HBase architecture

Data Storage & Operations

Each Region Server stores two types of files - a store file (HFile) and a write-ahead log (HLog). The HFile contains a variable number of data blocks and the fixed blocks for file information and trailer. Each data block contains a magic number and multiple key-value pairs. The default size of a data block is 64 KB. The index block stores the offset for the data and the meta-blocks. The trailer stores pointers to other blocks. HFiles are persisted on HDFS.

Each Region Server has a Memstore and a Block Cache. The Memstore stores the recent edits to the data in memory and the Block Cache caches the data blocks. Each Region Server also maintains a write-ahead log (WAL) known as the Hlog which logs the writes (that are also written to Memstore). Since HLog is stored on HDFS, it ensures that even in the event of loss of Memstore (which is an in-memory buffer), the writes are never lost.

Each Region Server has a Block cache, which is an in-memory store that caches the most recently used blocks for fast lookup.

HBase supports the following operations:

- Get: Get operation is used to return values for a given row key.

- Scan: Scan operation returns values for a range of row keys.
- Put: Put operation is used to add a new entry.
- Delete: Delete operation adds a special marker called Tombstone to an entry. Entries marked with Tombstones are removed during the compaction process

The storage structure used by HBase is a Log Structured Merge (LSM) Tree. LSM Tree uses two trees, one which is a smaller in-memory tree (in Memstore) and the other is a larger tree that is persisted on the disk (as store files). When the size of the in-memory tree exceeds a certain limit, it is merged with the larger tree on the disk using merge sort algorithm and a new in-memory tree is created. LSM Tree enables HBase to handle fast random reads and writes for large amounts of data. LSM Tree achieves this by transforming random data access to sequential data access.

Read Path:
For read operations (get or scan) the client first contacts Zookeeper to get the location of the ROOT table. The client then checks the ROOT table for correct META table containing the row key and obtains the Region Server name that is responsible for serving requests for that row-key. The client then contacts the Region Server directly to complete the read operation. The ROOT and META table lookups are cached by the client so that in subsequent read operations the client can directly contact the correct Region Server for a given row-key.

Write Path
All write requests are first logged into the WAL (HLog) sequentially. Once data is logged, it is also written to the Memstore. The Memstore stores the most recent updates to enable fast lookups. Over time, the Memstore starts filling up as new updates are stored. When the Memstore is filled up, it is flushed to the disk creating a new store file (HFile).

Compactions
Every time the Memstore is flushed to the disk, a new store file is created. Over time, many small store files are created on HDFS. Since HDFS is designed to work better with a smaller number of large files (as opposed to a large number of small files), a process called compaction is performed to merge the small files into a single file. The compaction process improves the read efficiency as a large number of small files don’t need to be looked up. HBase compactions are of two types - minor and major. Minor compaction merges the small files into a single file when the number exceeds a threshold. Minor compactions are done on a regular basis (typically multiple times in a day). Major compactions merge all store files into a single large store file. In the major compaction process, the outdated and deleted values (cells which have expired versions or cells marked with Tombstones) are removed. Compaction process improves the performance of HBase as looking up a single large store file for a get or scan operation is more efficient than looking up multiple small store files.

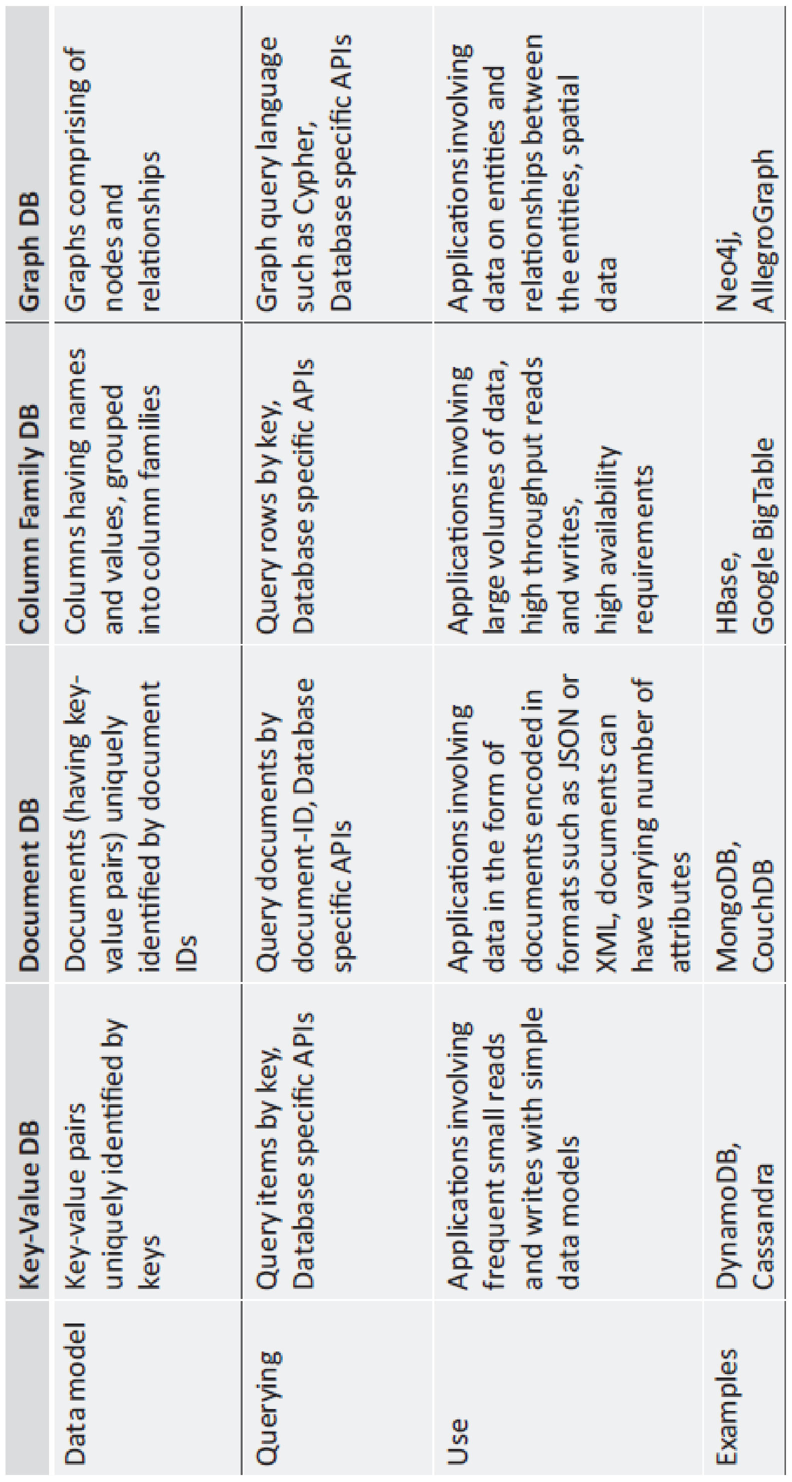
6.4 Graph Databases:
Graph stores are NoSQL databases designed for storing data that has graph structure with nodes and edges. While relational databases model data in the form of rows and columns, the graph databases model data in the form of nodes and relationships. Nodes represent the entities in the data model. Nodes have a set of attributes. A node can represent different types of entities, for example, a person, place (such as a city, restaurant or a building) or an object (such as a car). The relationships between the entities are represented in the form of links between the nodes. Links also have a set of attributes. Links can be directed or undirected. Directed links denote that the relationship is unidirectional. For example, for two entities

author and book, a unidirectional relationship called ‘writes’ exists between them, such that an author writes a book. Whereas for two friends, say A and B, the friendship relationship between A and B is bidirectional. In the graph theory terminology, the vertices in a graph are the nodes representing the entities and the edges between the vertices are the links between the nodes representing the relationships between the entities. A set of nodes along with the links between them form a path.

Graph databases, in contrast to relational databases, model relationships in the form of links between the nodes. Since the relationships between the entities are explicitly stored in the form of links, querying for related entities in graph databases is much simpler and faster than relational databases as the complex join operations are avoided. Graph databases are suitable for applications in which the primary focus is on querying for relationships between entities and analyzing the relationships.

Neo4j

Neo4j is one the popular graph databases which provides support for Atomicity, Consistency, Isolation, Durability (ACID). Neo4j adopts a graph model that consists of nodes and relationships. Both nodes and relationships have properties which are captured in the form of multiple attributes (key-value pairs). Nodes are tagged with labels which are used to represent different roles in the domain being modelled.



	Key-Value DB	Document DB	Column Family DB	Graph DB
Data model	Key-value pairs uniquely identified by keys	Documents (having key-value pairs) uniquely identified by document IDs	Columns having names and values, grouped into column families	Graphs comprising of nodes and relationships
Querying	Query items by key, Database specific APIs	Query documents by document-ID, Database specific APIs	Query rows by key, Database specific APIs	Graph query language such as Cypher, Database specific APIs
Use	Applications involving frequent small reads and writes with simple data models	Applications involving data in the form of documents encoded in formats such as JSON or XML, documents can have varying number of attributes	Applications involving large volumes of data, high throughput reads and writes, high availability requirements	Applications involving data on entities and relationships between the entities, spatial data
Examples	DynamoDB, Cassandra	MongoDB, CouchDB	HBase, Google BigTable	Neo4j, AllegroGraph

Figure 4.14: Comparison of NoSQL databases

7. SQL Vs. NOSQL:

SQL	NoSQL
Relational Database	Non Relational, distributed Database

Predefined schema	Dynamic schema for unstructured data
Vertically scalable	Horizontally scalable
Uses SQL	Uses UnQL(Unstructured Query Language)
Not preferred for large data sets	Preferred for large data sets
Emphasis on ACID properties	Follows CAP theorem
Supports complex querying	Does not have good support for complex querying
Examples: Oracle, MYSQL, MS SQL	Examples: MongoDB, Cassandra, CouchDB, Neo4j

Chapter 4
Introduction to Hadoop

4.1 Introduction

Hadoop plays an integral part in almost all big data processes. Hadoop is a framework that allows for distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. In other words, Hadoop is a software library that allows its users to process large datasets across distributed clusters of computers thereby enabling them to gather, store and analyze huge data sets. Hadoop provides various tools and technologies, collectively known as Hadoop ecosystem, to enable development and deployment of big data solutions.

4.2 Features of Hadoop:

Hadoop can handle large volumes of structured and unstructured data more efficiently than the traditional enterprise data warehouse. The power of Hadoop lies in its distributed processing. Hadoop is open source and therefore, it can run on commodity hardware. That means the initial cost savings are dramatic with Hadoop while it can continue to grow as your organizational data grows.
Here are a few key features of Hadoop:

4.2.1. Hadoop Brings Flexibility In Data Processing:

One of the biggest challenges organizations have had in that past was the challenge of handling unstructured data. Let's face it, only 20% of data in any organization is structured while the rest is all unstructured whose value has been largely ignored due to lack of technology to analyze it.
Hadoop manages data whether structured or unstructured, encoded or formatted, or any other type of data. Hadoop brings the value to the table where unstructured data can be useful in decision making process.

4.2.2. Hadoop Is Easily Scalable

This is a huge feature of Hadoop. It is an open source platform and runs on industry-standard hardware. That makes Hadoop extremely scalable platform where new nodes can be easily added in the system as and data volume of processing needs grow without altering anything in the existing systems or programs.

4.2.3. Hadoop Is Fault Tolerant

In Hadoop, the data is stored in HDFS where data automatically gets replicated at two other locations. So, even if one or two of the systems collapse, the file is still available on the third system at least. This brings a high level of fault tolerance. The level of replication is configurable and this makes Hadoop incredibly reliable data storage system. This means, even if a node gets lost or goes out of service, the system automatically reallocates work to another location of the data and continues processing as if nothing had happened.

4.2.4. Hadoop Is Great At Faster Data Processing

While traditional ETL and batch processes can take hours, days, or even weeks to load large amounts of data, the need to analyze that data in real-time is becoming critical day after day. Hadoop is extremely good at high-volume batch processing because of its ability to do parallel processing. Hadoop can perform batch processes 10 times faster than on a single thread server or on the mainframe.

4.2.5. Hadoop Ecosystem Is Robust:

Hadoop has a very robust ecosystem that is well suited to meet the analytical needs of developers and small to large organizations. Hadoop Ecosystem comes with a suite of tools and technologies making it very much suitable to deliver to a variety of data processing needs. Just to name a few, Hadoop ecosystem comes with projects such as MapReduce, Hive, HBase, Zookeeper, HCatalog, Apache Pig etc. and many new tools and technologies are being added to the ecosystem as the market grows.

4.2.6. Hadoop Is Very Cost Effective:

Hadoop generates cost benefits by bringing massively parallel computing to commodity servers, resulting in a substantial reduction in the cost per terabyte of storage, which in turn makes it reasonable to model all your data. Apache Hadoop was developed to help Internet-based companies deal with prodigious volumes of data. According to some analysts, the cost of a Hadoop data management system, including hardware, software, and other expenses, comes to about $1,000 a terabyte–about one-fifth to one-twentieth the cost of other data management technologies

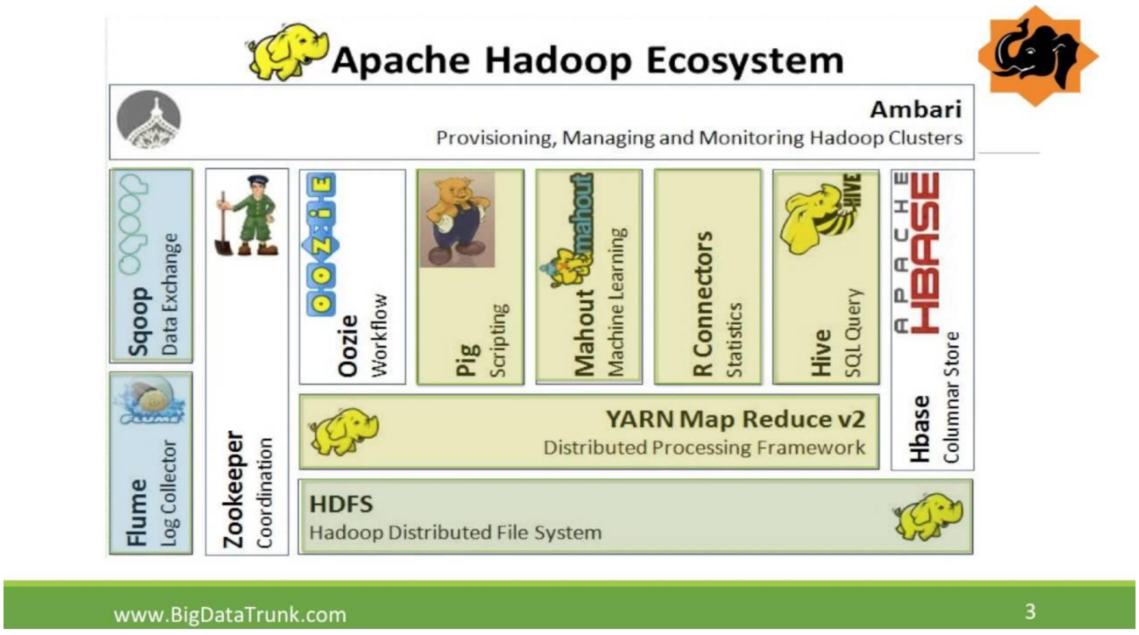
4.3 Hadoop Ecosystem:

Hadoop ecosystem is a framework of various types of complex tools and components. These elements may be very different from each other in terms of their architecture, but, they all derive their functionality from the scalability and power of Hadoop. Hadoop ecosystem can thus be defined as a comprehensive collection of tools and technologies that can be effectively implemented and deployed to provide big data solutions in a cost effective manner. MapReduce and Hadoop Distributed File System (HDFS) are the two core components of Hadoop ecosystem that help to manage Big Data.

4.3.1 Elements of Hadoop Ecosystem:

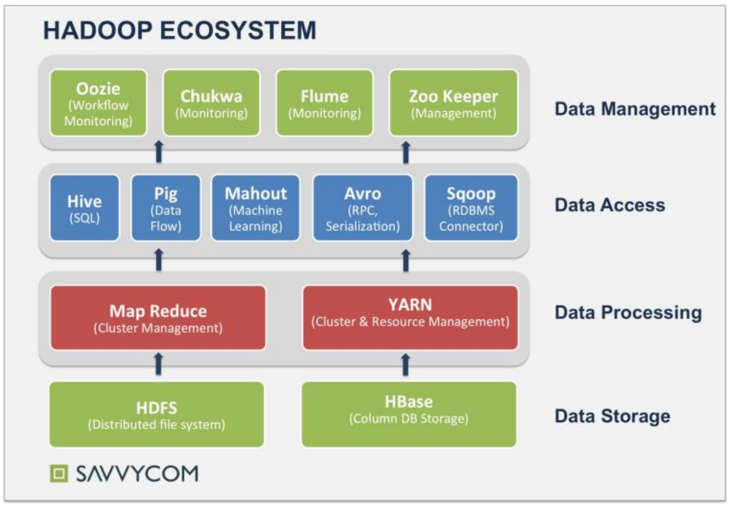
The idea of a Hadoop ecosystem involves the use of different parts of the core Hadoop set such as MapReduce, a framework for handling vast amounts of data, and the Hadoop Distributed File System (HDFS), a sophisticated file-handling system. There is also YARN, a Hadoop resource manager.
In addition to these core elements of Hadoop, Apache has also delivered other kinds of accessories or complementary tools for developers. These include Apache Hive, a

data analysis tool; Apache Spark, a general engine for processing big data; Apache Pig, a data flow language; HBase, a database tool; and also Ambari, which can be considered as a Hadoop ecosystem manager, as it helps to administer the use of these various Apache resources together. With Hadoop becoming the de facto standard for data collection and becoming ubiquitous in many organizations, managers and development leaders are learning all about the Hadoop ecosystem and what kinds of things are involved in a general Hadoop setup.



4.3.3 Layers of Hadoop Ecosystem:

Hadoop Ecosystem Elements with Various Stages of data processing



HADOOP ECOSYSTEM
Oozie (Workflow Monitoring)
Chukwa (Monitoring)
Flume (Monitoring)
Zoo Keeper (Management)
Data Management
Hive (SQL)
Pig (Data Flow)
Mahout (Machine Learning)
Avro (RPC, Serialization)
Sqoop (RDBMS Connector)
Data Access
Map Reduce (Cluster Management)
YARN (Cluster & Resource Management)
Data Processing
HDFS (Distributed file system)
HBase (Column DB Storage)
Data Storage
SAVVYCOM

Data Storage Layer:

- **HDFS** (Hadoop Distributed File System) is the key component that makes up Hadoop. HDFS is used to store and access huge file based on client/server architecture. This system also enables the distribution and storage of data across Hadoop clusters.

- **HBase** (Hadoop Database) is a columnar database built on top of the HDFS. Being a file system, HDFS lacks the random read and write capability. It is when HBase steps in and provides fast record lookups in large tables.

Data Processing Layer:

- **MapReduce** is a parallel data processing framework over clusters. Using MapReduce can help data seeker save a lot of time, for example, if it takes a normal relational database around 20 hours to process a large data set, it might take MapReduce only around three minutes to get everything done.
- **YARN** (Yet Another Resource Negotiator) is a resource manager. It is said to be the second generation of MapReduce and also a critical advancement from Hadoop 1. YARN acts the role of an operating system, its jobs is to manage and monitor workloads, make sure it can serve multiple clients and perform security controls. In addition, YARN supports new processing models that MapReduce does not.

Data Access Layer

- **Hive** is new kind of structured query language. It was born to help who are familiar with the traditional database and SQL to leverage Hadoop and MapReduce.
- **Pig** serves the analysis purpose for large data sets. Pig is made up of two components, firstly the platform to execute Pig programs; secondly, a powerful and simple scripting language called PigLatin, which is used to write those programs.
- **Mahout** provides a library of the most popular machine learning algorithms written in Java that supports collaborative filtering, clustering, and classification.
- **Arvo** is a data serialization system. It uses JSON for defining data types and protocols to support data-driven applications. Arvo provides a simple integration with many different languages with the expectation to support Hadoop application to be written in other languages (e.g. Python, C++) rather than Java.
- **Sqoop** (SQL + Hadoop = Sqoop) is a command line interface application, which helps transfer data between Hadoop and relational databases (e.g. MySQL or Oracle) or mainframes.

Data Management Layer:

- **Oozie** is a workflow scheduler for Hadoop. Oozie streamlines the process of creating workflows and managing coordination jobs among Hadoop and other applications such as Map Reduce, Pig, Sqoop, Hive etc. The main responsibilities of Oozie are: firstly to define a sequence of actions to be executed; secondly, to place triggers for those actions.
- **Chukwa** is another framework that is built on top of HDFS and Map Reduce. Its purpose is to provide a dynamic and powerful data collection system. Chukwa is capable of monitoring, analyzing and presenting the results to get the most out of collected data.
- **Flume** is also a scalable and reliable system for collecting and moving cluster logs from various sources to a centralized store like Chukwa. However, there are some differences. In Flume, chunks of data are transferred from node to node in store and

forward manner; while in Chukwa, the agent of each machine will need to determine what data to be sent.

- **ZooKeeper** is a distributed coordination service for distributed system. It provides a very simple programming interface and helps reduce the management complexity by providing services such as configuration, distributed synchronization, naming, group services etc.

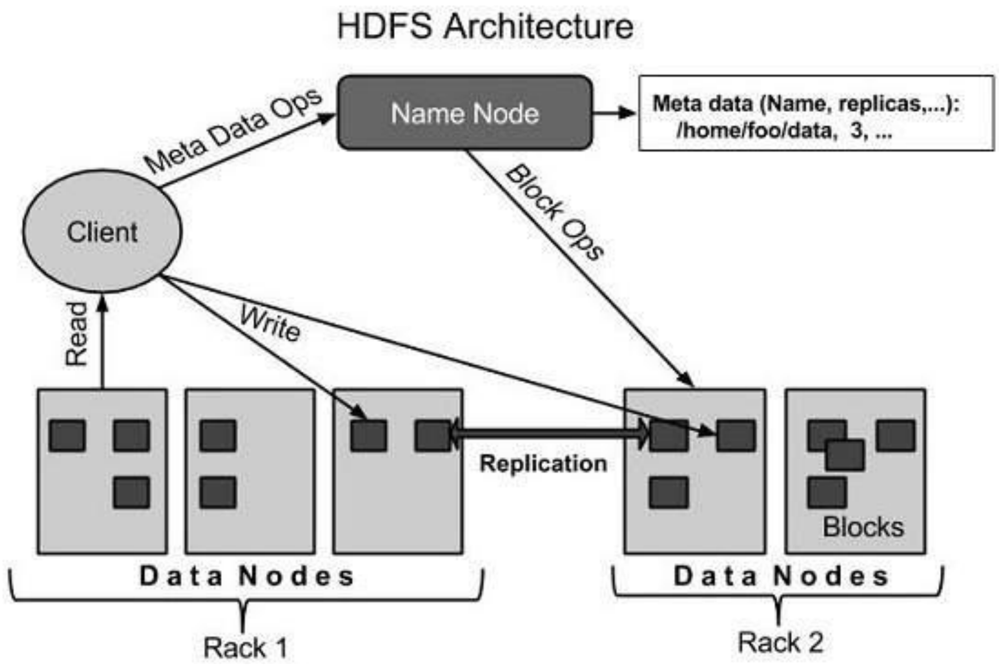
4.4 HDFS Architecture:

HDFS follows the master-slave architecture. It has a NameNode and a number of DataNodes. The NameNode is a master that manages the various DataNodes.

NameNode

The NameNode is the commodity hardware that contains the GNU/Linux operating system and the NameNode software. It is a software that can be run on commodity hardware. The NameNode manages the HDFS cluster metadata. The records and directories are managed on the NameNode. Operations on them, such as modification, opening or closing are performed by NameNode . The system having the NameNode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.



DataNode:

The DataNode is a commodity hardware having the GNU/Linux operating system and DataNode software. For every node (Commodity hardware/System) in a cluster, there will be a DataNode. These nodes manage the data storage of their system. DataNodes perform read-write operations on the file systems, as per client request. They also perform operations such as block creation, deletion, and replication according to the instructions of the NameNode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

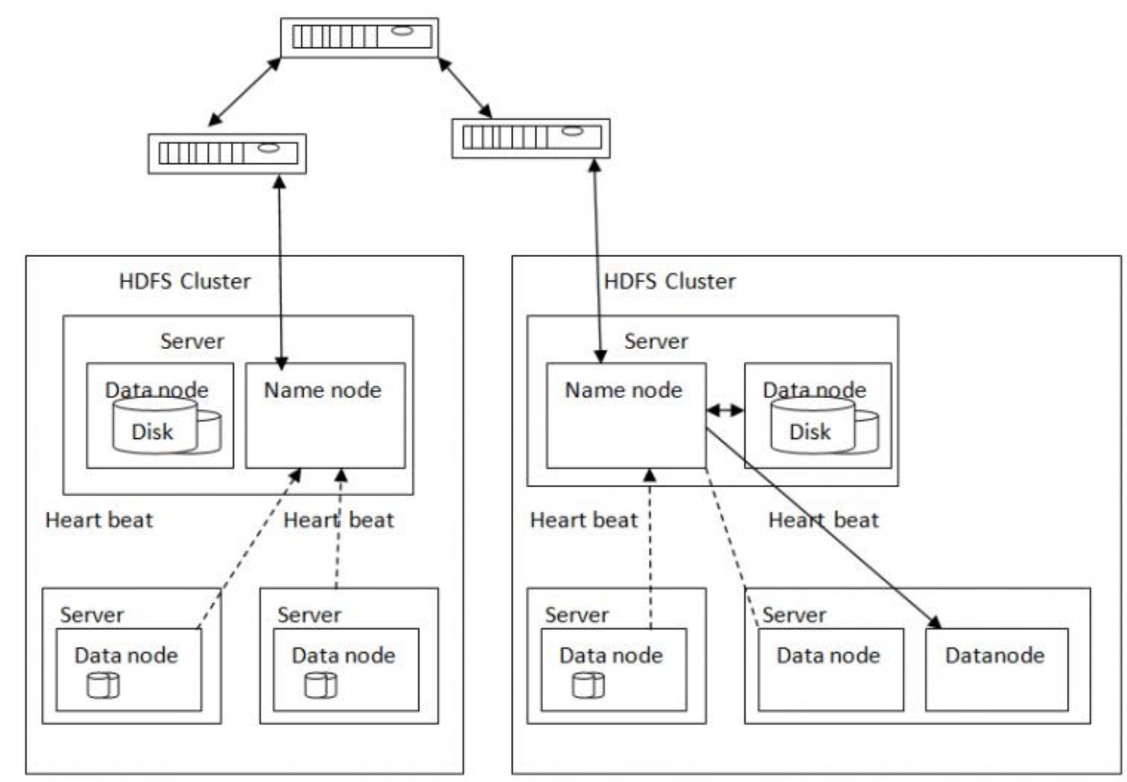
Hadoop Heartbeat Message:

The important objective of HDFS is to store data reliably, even when features occur with Name Nodes, data nodes or network partitions.
Detection is, the first step HDFS takes to overcome failures and HDFS uses heart beat messages to detect connectivity between home and data nodes

Hadoop Heartbeat

Several things can cause loss of connectivity between name and data nodes and therefore each data node sends periodic heartbeat messages to its NameNodes so the latter can detect loss of connectivity if it stops receiving them.The Name Node marks as dead data nodes not responding to heart beats and refrains from sending further requests to them. Data stored on a data node is no longer available to an HDFS client from that node, which is effectively removed from the system. If the death of a node causes the replication factor of data blocks to drop below their minimum value, the NameNode initiates additional replication to normalized state.

Heart Beat Process Diagram



4.5 Features of HDFS

4.5.1. Fault Tolerance

Fault tolerance in **HDFS** refers to the working strength of a system in unfavorable conditions and how that system can handle such situations. HDFS is highly fault-tolerant, in HDFS data is divided into blocks and multiple copies of blocks are created on different machines in the cluster (this replica creation is configurable). So whenever if any machine in the cluster goes down, then a client can easily access their data from the other machine which contains the same copy of data blocks. HDFS also maintains the replication factor by creating a replica of blocks of data on another rack. Hence if suddenly a machine fails, then a user can access data from other slaves present in another rack. To learn more about Fault Tolerance follow this Guide.

4.5.2. High Availability

HDFS is a highly available file system, data gets replicated among the nodes in the HDFS cluster by creating a replica of the blocks on the other slaves present in HDFS cluster. Hence whenever a user wants to access this data, they can access their data from the slaves which contains its blocks and which is available on the nearest node in the cluster. And during unfavorable situations like a failure of a node, a user can easily access their data from the other nodes. Because duplicate copies of blocks which contain user data are created on the other nodes present in the HDFS cluster. To learn more about high availability follow this Guide.

4.5.3. Data Reliability

HDFS is a distributed file system which provides reliable data storage. HDFS can store data in the range of 100s of petabytes. It also stores data reliably on a cluster of nodes. HDFS divides the data into blocks and these blocks are stored on nodes present in HDFS cluster. It stores data reliably by creating a replica of each and every block present on the nodes present in the cluster and hence provides fault tolerance facility. If node containing data goes down, then a user can easily access that data from the other nodes which contain a copy of same data in the HDFS cluster. HDFS by default creates 3 copies of blocks containing data present in the nodes in HDFS cluster. Hence data is quickly available to the users and hence user does not face the problem of data loss. Hence HDFS is highly reliable.

4.5.4. Replication

Data Replication is one of the most important and unique features of Hadoop HDFS. In HDFS replication of data is done to solve the problem of data loss in unfavorable conditions like crashing of a node, hardware failure, and so on. Since data is replicated across a number of machines in the cluster by creating blocks. The process of replication is maintained at regular intervals of time by HDFS and HDFS keeps creating replicas of user data on different machines present in the cluster. Hence whenever any machine in the cluster gets crashed, the user can access their data from other machines which contain the blocks of that data. Hence there is no possibility of losing of user data. Follow this guide to learn more about the data read operation.

4.5.5. Scalability

As HDFS stores data on multiple nodes in the cluster, when requirements increase we can scale the cluster. There is two scalability mechanism available: Vertical scalability – add more resources (CPU, Memory, Disk) on the existing nodes of the cluster. Another way is horizontal scalability – Add more machines in the cluster. The horizontal way is preferred since we can scale the cluster from 10s of nodes to 100s of nodes on the fly without any downtime.

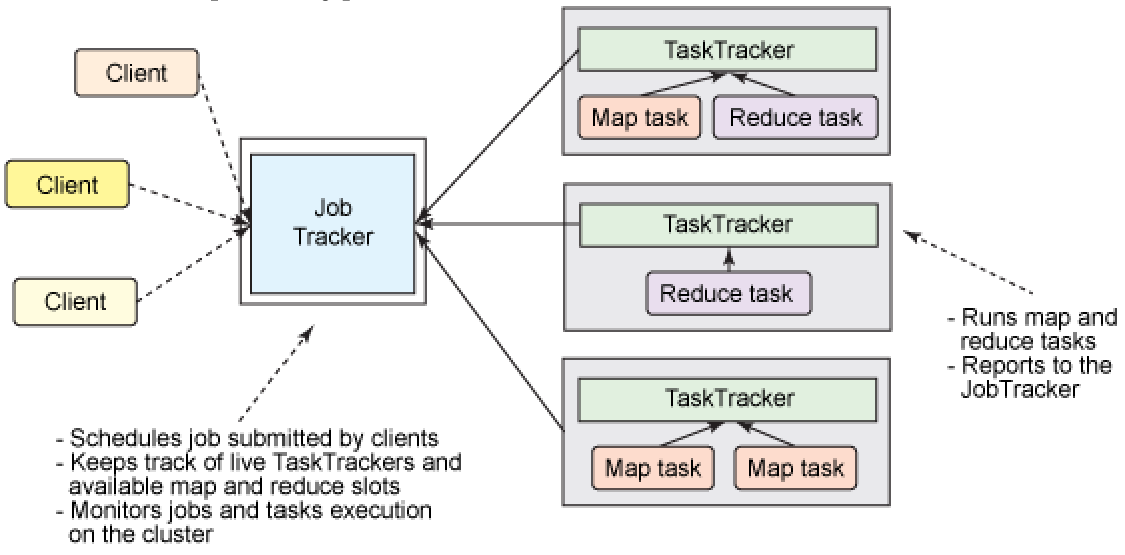
4.5.6. Distributed Storage

In HDFS all the features are achieved via distributed storage and replication. HDFS data is stored in distributed manner across the nodes in HDFS cluster. In HDFS data is divided into blocks and is stored on the nodes present in HDFS cluster. And then replicas of each and every block are created and stored on other nodes present in the cluster. So if a single machine in the cluster gets crashed we can easily access our data from the other nodes which contain its replica.

4.6 Hadoop YARN:

4.6.1 History of YARN:

The data processing component is separated from the resource management and scheduling components of MapReduce. This helps Hadoop ecosystem to run interactive queries and streaming data applications simultaneously through MapReduce batch jobs. It helps Hadoop to support broader range of applications and different types of processing approaches. In this environment, we can run various user applications of different types on multiple platforms simultaneously.
Hadoop 1 with core MapReduce processing provides a way in which MapReduce programs could be written in various languages such as Java, Python, Ruby on Rails etc. The issue with Hadoop 1 is that programs are run on the basis of MapReduce processing cannot deal with all big data processing problems. The MapReduce processing model is a program based model which includes tools like Pig and Hive to provide simplicity to data processing tasks. YARN was introduced to fill the gap resulted due to the incapability of MapReduce model to fulfill all kinds of data processing requirements
MapReduce is a programming model to process large amounts of data. It is a part of Apache Hadoop. It divides large data sets into multiple small datasets distributed over a number of mapper and reducer nodes used for processing data. The MapReduce consists of a JobTracker node that is also called the master and multiple TaskTracker nodes known as slaves.
The JobTracker performs the resource management tasks which include managing the slaves, tracking the availability and consumption of resources and managing the job life cycle. The management of job lifecycle involves individual task scheduling, progress tracking and providing fault tolerance. TaskTracker nodes, on the other hand, perform fundamental operations that include breaking down of tasks on the orders of JobTracker and providing periodic status information to the Jobtracker.


Client
Client
Client
Job Tracker
TaskTracker
Map task
Reduce task
TaskTracker
Reduce task
TaskTracker
Map task
Map task
- Runs map and reduce tasks
- Reports to the JobTracker
- Schedules job submitted by clients
- Keeps track of live TaskTrackers and available map and reduce slots
- Monitors jobs and tasks execution on the cluster

4.6.2 Advantages of YARN:

In Hadoop, the task of cluster management is handled by YARN and the tasks of data processing are handled by using MapReduce programming paradigm. YARN manages the cluster through a component called central resource manager, which manages the resources and allocates them to applications. The inclusion of YARN in Hadoop 1 eliminates the need of JobTracker and TaskTracker. Some of the advantages of YARN are listed below

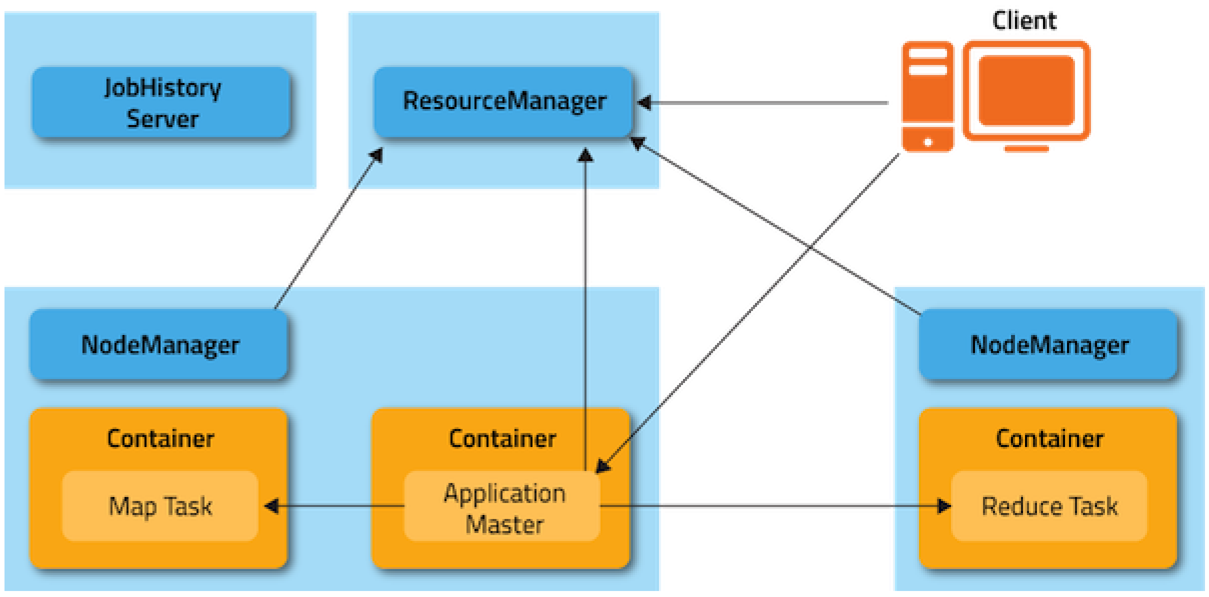
- YARN provides efficient resource utilization. The available resources are scheduled to perform map and reduce tasks per requirement instead of prior allocation of resources for map and reduce job execution
- Multiple applications that share common resources can be run simultaneously.
- The resource management and scheduling tasks of MapReduce are separated from data processing component by YARN. Hence, a large variety of data processing tasks and wide range of applications are supported by YARN
- YARN provides backward compatibility i.e. applications developed on Hadoop 1 can run on Hadoop 2 directly
- Managing of resources and monitoring of jobs are assigned to two components called resource manager and application manager

4.6.3 YARN Architecture:

The two primary components of YARN architecture are Resource Manager and Application Manager.

a. ResourceManager – ResourceManager in YARN is the supreme authority that controls all the decisions related to resource management and allocation. It has a scheduler API that negotiates and schedules the resources. The scheduler API does not monitor or track the status of applications. This API schedules the resources such as CPU, memory, disk and network on the basis of the resource requirements of the applications. YARN ResourceManager is responsible for almost all the tasks performed by JobTracker in MapReduce model. ResourceManager perform all the tasks in integration with the ApplicationManager and NodeManager. Resources on each node are allocated and managed by its respective NodeManager. The ResourceManager gives instructions to the NodeManager, which is responsible for managing the resources available on the node it manages. For each application, there is an ApplicationManager instance that negotiates the resources with the ResourceManager. The application in association with the NodeManager instances starts the container. Containers are physical resources like memory, CPU and disks that are available at every node.

Architectural view of YARN



b. ApplicationManager – Every instance of an application running on YARN is managed by an ApplicationManager. It is responsible for negotiation of resources with ResourceManager. ApplicationManager also keep track of the availability and consumption of container resources and provides fault tolerance for resources. ApplicationManager is responsible for negotiaying of appropriate resource containers from scheduler, monitoring their status and checking their progress.

c. Containers – A container is nothing but a set of physical resources on a single node. It contains of memory, CPU and disks. Depending on the resources in the node, a node can have multiple containers that are assigned to specific ApplicationManager. A container represents a resource on a single node on a given cluster. It is supervised by a NodeManager and supervised by a ResourceManager

d. NodeManager – It is a per-machine slave, which is responsible for launching the application containers and reporting the status of resource usage to the ResourceManager. NodeManager manages each node with a YARN cluster. It provides per-node services within the cluster. These services range from managing a container to monitoring resources and tracking the health of its node.

4.6.4 YARN Schedulers:

There are two types of schedulers commonly available with YARN for negotiating resources with the ApplicationManager via ResourceManager. These Schedulers are CapacitySchedulers and FairSchedulers.

a. CapacityScheduler

It is the default scheduler available in Hadoop 2. Its purpose is to allow multi-tenancy and share resources between multiple organizations and applications on the same cluster. The job of CapacityScheduler is to make sure that any user, application should not consume a lot of resources available in a cluster. By applying limitation on the initialized and pending application from a single user they ensure stability and fairness among clusters.

The features of CapacityScheduler are

- Hierarchical queues – CapacityScheduler supports hierarchy of queues. This hierarchy ensures that the resources are equally shared among the sub queues of an organization before other queues are allowed to use the free resources. It provides better control and predictability to the application.
- Capacity guarantees – Every application that is submitted to a queue can access the entire capacity of resources allocated to the queue by the administrators.
- Security – Every organization has a unique queue. Users of the queues cannot modify or delete the application or properties belonging to other queues.
- Elasticity – In certain cases, free resource can be allocated to any queue beyond its capacity. CapacityScheduler ensures that resources are available to the queues in a predictable and elastic manner.
- Multi-tenancy – CapacityScheduler does not provide a comprehensive set of limits to a single application, user and queue. This prevents over utilization of resources.
- Resource-based Scheduling – CapacityScheduler supports resource intensive applications. These applications specify higher resource requirements than the default.

b. FairScheduler

FairScheduling is a method of assigning resources to applications via ApplicationManager such that all applications get an equal share of resources during their course of running. YARN is capable of scheduling various types of multiple resources. By default, FairScheduler schedules memory; however it can be configured to schedule both memory and CPU. When there is a single application running in the cluster, the ApplicationManager uses the entire cluster; but when more than one application is running, then the resources that are free can be assigned to the new application.

Assignment:

1. Explain briefly the various types of NoSQL models
2. Compare and contrast SQL and NOSQL
3. Explain the working of DynamoDB databases.
4. Explain the working of HBase along with its features
5. Explain the advantages of NoSQL.
6. Explain the components of Hadoop
7. List and explain the features of hadoop
8. Explain the hadoop heartbeat process with a neat diagram
9. Explain the YARN architecture with a neat diagram
10. Explain the various types of YARN scheduler.

Essay Questions(10 marks)

1. Explain the HBase architecture with a neat diagram
2. Explain the layers of hadoop ecosystem with a neat diagram
3. Explain the HDFS architecture with a neat diagram