UNIT III

3 Big Data Processing

Syllabus

Big Data Analytics - Ecosystem and Technologies, *Introduction to Google file system, Hadoop Architecture,* ***Hadoop Storage :*** *HDFS, Common Hadoop Shell commands, Anatomy of File Write and Read, NameNode, Secondary NameNode, and DataNode, Hadoop MapReduce paradigm, Map Reduce tasks, Job, Task trackers - Cluster Setup - SSH & Hadoop Configuration, Introduction to NOSQL, Textual ETL processing.*

Contents

3.1 Big Data Ecosystem

SPPU : May-18

- Big data ecosystem is the comprehension of massive functional components with various enabling tools. Capabilities of the big data ecosystem are not only about computing and storing big data, but also the advantages of its systematic platform and potentials of big data analytics.
- Organizations and data collectors that can gather data from individuals start realising that a new economy is emerging as data business. Depending on the evolution of this economy, a new ecosystem is rising.
- Many organizations and data collectors are depends on the data they can gather from individuals contains great value and as a result, a new economy is emerging.
- This new digital economy continues to evolve; the market sees the introduction of data vendors and data cleaners that use crowd sourcing to test the outcomes of machine learning techniques.
- As the ecosystem has been growing, groups of interest have been formed. Currently there are four main group which are associated with each other : Data devices, data collectors, data aggregators and data users and buyers.
- Fig. 3.1.1 shows emerging big data ecosystem.

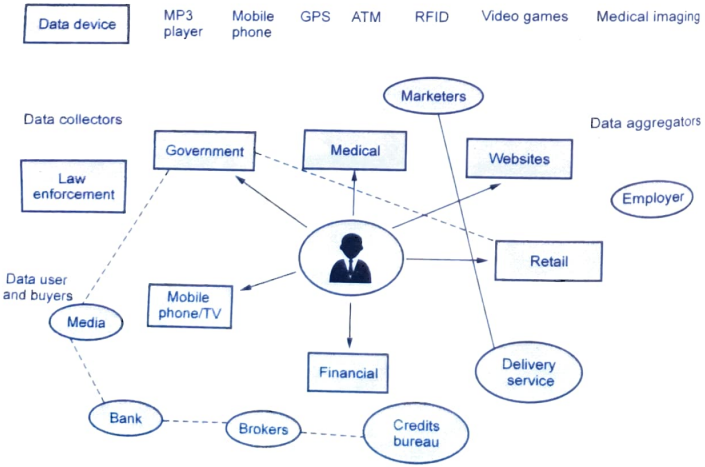

Fig. 3.1.1 Emerging big data ecosystem

1. **Data device :**
   - Data device and sensor network gather data from various locations and continuously generate new data.
   - **Example of data devices :** Playing games, smart phone and retail **shopping.**
   - **Sensor data :** Growing network of sensor devices generate data based on monitoring environmental conditions, such as temperature, sound, pressure, power, water level etc.
   - This data can have a wide range of practical applications if collected, aggregated, analyzed and acted upon. Examples include, water level monitoring, machine health monitoring and smart home monitoring.
   - **Mobile networks :** Mobile network generates large number of data to share picture, video, audio file and text. These data is process at every mobile tower with associated demographics, location latencies etc. Sometime, mobile network may crash for large number of data movement takes place.
   - Retail shopping loyalty cards records not just the amount an individual spends, but the location of stores that person visits, the kinds of product purchased, the store where goods are purchased most often and the combinations of product purchased together.
2. **Data collectors :**
   - Data collectors : It includes sample entities that collect data from the device and user.
   - Retail stores tracking the path a customer takes through their store while pushing a shopping cart with an RFID chip so they can gauge which products get the most foot traffic using geospatial data collected from the RFID chips.
   - Data collectors example : Government, Retail stores, Cable TV provider.
   - Cable TV provider which tracks the shows that a person watches.
3. **Data aggregators :**
   - The entities which process collected data from the first layer make them understandable. They give them additional value to prepare them for the handing over process. Now the data is ready to be offered on the market.
   - Typically, one of these data aggregations can transform and package the data as products to sell to list brokers that might want to generate marketing lists of people who may be good targets for specific ad campaigns.
4. **Data users and buyers :**
   - These entities represent a group of the final layer from the Big Data ecosystem. This group has the final benefits from the collected and aggregated data offered by the data aggregations.

- Data users may want to track or prepare for natural disasters by identifying which areas a hurricane will affect first hand. It can be observed by tracking tweets about it or discussing it in social media.

Review Question

1. *What is the need of big data analysis ? Explain the different types of analysis techniques.* **SPPU : May-18 (End Sem), Marks 8**

3.2 Introduction to Google File System

SPPU : May-18, April-19, 20, Dec.-19

- Google file system is "a scalable distributed file system for large distributed data-intensive applications" created by Google. Initially used to store Google's search indexes and the crawling data, GFS is now mostly used to store user generated content.
- GFS was built primarily as the fundamental storage service for Google's search engine.
- GFS typically will hold a large number of huge files, each 100 MB or larger, with files that are multiple GB in size quite common. Thus, Google has chosen its file data block size to be 64 MB instead of the 4 KB in typical traditional file systems. The I/O pattern in the Google application is also special.
- Files are typically written once, and the write operations are often the appending data blocks to the end of files. Multiple appending operations might be concurrent. There will be a lot of large streaming reads and only a little random access.
- There is no data cache in GFS as large streaming reads and writes represent neither time nor space locality

Motivation behind GFS

- Fault tolerance and auto-recovery need to be built into the systems i.e. monitoring, error detection, fault tolerance, automatic recovery. Because problems are very often caused by application bugs, OS bugs, human errors, and the failure of disks, memory, connectors, networking, and power supplies.
- Standard I/O assumptions (e.g. block size) have to be re-examined.
- Record appends are the frequent form of writing and Google applications and GFS should be co-designed.

3.2.1 GFS Architecture

- A GFS cluster consists of a single master and multiple chunk servers and is accessed by multiple clients.

Basic terms :

a. **Master :** Single, coordinates system-wide activities. Can have read-only 'Shadow' servers.

b. **Chunk :** 64 MB storage block representing a file or piece thereof.

c. **Chunkserver :** Many, stores chunks of data.

d. **Replica :** Either primary or secondary. A Chunkserver that replicates a given block.

e. **Client :** Runs tasks on data.

- It is easy to run both a chunkserver and a client on the same machine, as long as machine resources permit.
- Files are divided into fixed-size chunks. Each chunk is identified by an immutable and globally unique 64 bit chunk handle assigned by the master at the time of chunk creation.
- Chunkservers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers.
- The master maintains all file system metadata. This includes the namespace, access control information, the mapping from files to chunks and the current locations of chunks.
- Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers.
- Neither the client nor the chunkserver caches file data. Fig. 3.2.1 shows GFS architecture.

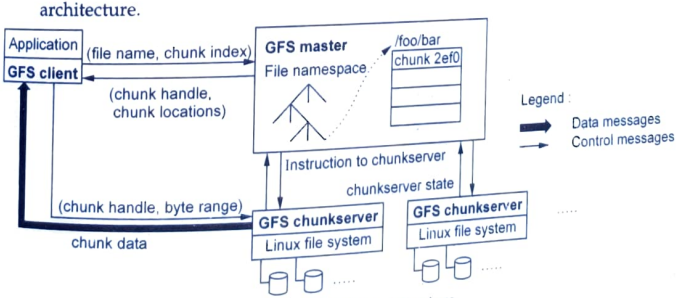

Fig. 3.2.1 GFS architecture

- Clients never read and write file data through the master. Instead, a client asks the master which chunkservers it should contact. It caches this information for a

limited time and interacts with the chunkservers directly for many subsequent operations.

- First, using the fixed chunk size, the client translates the file name and byte offset specified by the application into a chunk index within the file. Then, it sends the master a request containing the file name and chunk index.
- The master replies with the corresponding chunk handle and locations of the replicas. The client caches this information using the file name and chunk index as the key. The client then sends a request to one of the replicas, most likely the closest one.
- The request specifies the chunk handle and a byte range within that chunk.
- Further reads of the same chunk require no more client-master interaction until the cached information expires or the file is reopened.
- In fact, the client typically asks for multiple chunks in the same request and the master can also include the information for chunks immediately following those requested. This extra information sidesteps several future client-master interactions at practically no extra cost.

3.2.2 Chunk Size and Metadata of GFS

Chunk size :

- Chunk size is 64 MB. Each chunk replica is stored as a plain Linux file on a chunkserver and is extended only as needed.
- Advantages of large chunk :
  1. It reduces the network overheads.
  2. It reduces the size of the metadata stored on the master.
  3. It reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information.

Metadata :

- The master stores three major types of metadata :
  1. File and chunk namespaces,
  2. Mapping from files to chunks,
  3. Locations of each chunk's replicas.
- All metadata is kept in the master's memory. The first two types are also kept persistent by logging mutations to an operation log stored on the master's local disk and replicated on remote machines.

- Using a log allows us to update the master state simply, reliably and without risking inconsistencies in the event of a master crash.
- The master does not store chunk location information persistently. Instead, it asks each chunkserver about its chunks at master startup and whenever a chunkserver joins the cluster.
- Chunk replicas are created for three reasons : Chunk creation, re-replication, and rebalancing.

3.2.3 Data Mutation Sequence in GFS

- Here we discuss Write Control and Data Flow (data mutation) sequence of GFS.
- Fig. 3.2.2 shows process of control flow of a write through.

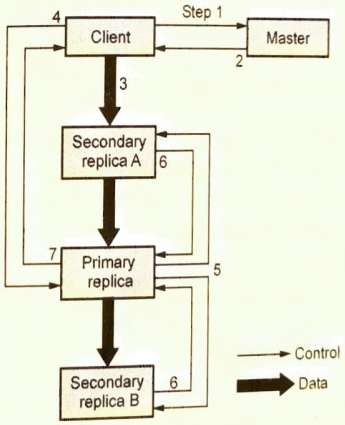

Fig. 3.2.2 Process of control flow of a write through

- The client asks the master which chunk server holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a replica it chooses.
- The master replies with the identity of the primary and the locations of the other (secondary) replicas. The client caches this data for future mutations. It needs to contact the master again only when the primary becomes unreachable or replies that it no longer holds a lease.

- The client pushes the data to all the replicas. A client can do so in any order. Each chunk server will store the data in an internal LRU buffer cache until the data is used or aged out.
- Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary.
- The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.
- The secondary's all reply to the primary indicating that they have completed the operation.
- The primary replies to the client. Any errors encountered at any of the replicas are reported to the client.

3.2.4 Big Data Processing Challenge

- Big data organizes and extracts the valued information from the rapidly growing large volumes, variety forms and frequently changing data sets collected from multiple and autonomous sources in the minimal possible time, using several statistical and machine learning techniques.
- GFS is high performance file system on commodity hardware clusters for large scale data processing.
- Google offers big query to operate on Google big tables. New generation of query languages, examples are Google big query.
- Web log mining is the study of the data available in the web. This involves searching for the texts, words, and their occurrences.
- One example for web log mining is searching for the words, and their frequencies by Google big query data analytics use Google big query platform to run on the Google cloud infrastructure.
- Mapreduce framework has made complex large-scale data processing easy and efficient. MapReduce is inherently designed for high throughput batch processing of big data that take several hours and even days, while recent demands are more centered on jobs and queries that should finish in seconds or at most, minutes.

Review Questions

1) *Explain how Google file system solves big data processing challenges.*
2) *Explain the following terms : 1) Google file system.* **SPPU : May-18 (End Sem), Marks 4**
3) *Explain different steps in data analytics project life cycle.* **SPPU : April-19 (In Sem), Marks 3**
4) *Explain the-GFS system with respect to i) Architecture ii) Types of metadata iii) Strengths.* **SPPU : Dec.-19 (End Sem), Marks 8**
SPPU : April-20 (In Sem), Marks 6

3.3 Hadoop Architecture

SPPU : April-18, 19, 20, May-18, Dec.-18, 19

- Hadoop Distributed File System (HDFS) is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.
- The Hadoop core is divided into two fundamental layers : The MapReduce engine and HDFS.
- The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.
- Hadoop is an open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license. It provide software framework for distributed processing of large datasets in real-time applications.
- Hadoop provides the basic platform for big data processing. The hadoop architecture have mainly two parts : Hadoop distributed File System (HDFS) and the MapReduce engine.
- HDFS is Distributed Files system designed to run on commodity hardware, which is highly fault- tolerant andscalable. Fig. 3.3.1 shows HDFS architecture.
- Hadoop Distributed File System is a block - structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines.
- Apache Hadoop HDFS Architecture follows a Master/Slave Architecture, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).
- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).

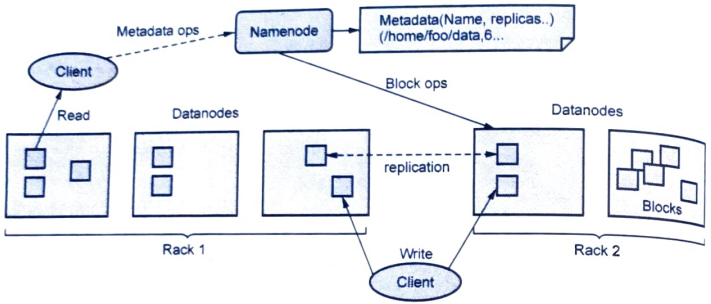

Fig. 3.3.1 Hadoop architecture

- HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.
- NameNode is the master node in the Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes).
- NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients.
- DataNodes are the slave nodes in HDFS. Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. The DataNode is a block server that stores the data in the local file ext3 or ext4.
- Journal is the modification log of image, which is available in local hosts native file system. Journal is updated for every client transaction.
- Checkpoint is persistent record of the image, which is also stored on local hosts native file system to enable recovery. NameNode is not allowed to update or modify Checkpoint file.
- Administrator or Checkpoint Node can demand to create new checkpoint file on startup, or restart.

3.3.1 Node Types

1. NameNode :

- It is also known as Master node. Namenode stores meta-data i.e. number of blocks, their location, replicas and other details.
- This meta-data is available in memory in the master for faster retrieval of data.
- NameNode maintains and manages the slave nodes and assigns tasks to them. It should be deployed on reliable hardware as it is the centerpiece of HDFS.

2. Checkpoint node :

- Checkpoint node is a node which periodically creates checkpoints of the namespace.
- Checkpoint Node in Hadoop first downloads fsimage and edits from the active Namenode. Then it merges them locally and at last it uploads the new image back to the active NameNode.
- The checkpoint Node stores the latest checkpoint in a directory. It is structured in the same as the Namenode's directory. It permits the checkpointed image to available for reading by the namenode.

3. Backup node :

- Backup node provides the same checkpointing functionality as the Checkpoint node. In Hadoop, Backup node keeps an in-memory, up-to-data copy of the file system namespace, which is always synchronized with the active NameNode state.
- The Backup node does not need to download fsimage and edits files from the active NameNode in order to create a checkpoint, as would be required with a Checkpoint node or Secondary Namenode, since it already has an up-to-data state of the namespace state in memory.
- The Backup node checkpoint process is more efficient as it only needs to save the namespace into the local fsimage file and reset edits.
- One Backup node is supported by the NameNode at a time. No checkpoint nodes may be registered if a Backup node is in use.

3.3.2 Block Placement in Hadoop

- For a large cluster, it may not be practical to connect all nodes in a flat topology. A common practice is to spread the nodes across multiple racks.
- Nodes of a rack share a switch and rack switches are connected by one or more core switches. Communication between two nodes in different racks has to go through multiple switches.
- In most cases, network bandwidth between nodes in the same rack is greater than network bandwidth between nodes in different racks.
- Fig. 3.3.2 shows cluster with two racks, each of which contains three nodes.

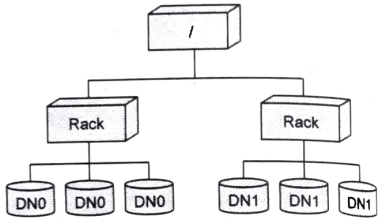

Fig. 3.3.2 Block placement

- HDFS estimates the network bandwidth between two nodes by their distance. The distance from a node to its parent node is assumed to be one.
- A distance between two nodes can be calculated by summing the distances to their closest common ancestor. A shorter distance between two nodes means greater bandwidth they can use to transfer data.
- HDFS allows an administrator to configure a script that returns a node's rack identification given a node's address.
- The NameNode is the central place that resolves the rack location of each DataNode. When a DataNode registers with the NameNode, the NameNode runs the configured script to decide which rack the node belongs to.
- If no such a script is configured, the NameNode assumes that all the nodes belong to a default single rack.
- The default HDFS block placement policy provides a tradeoff between minimizing the write cost and maximizing data reliability, availability and aggregate read bandwidth.
- When a new block is created, HDFS places the first replica on the node where the writer is located. The second and the third replicas are placed on two different nodes in a different rack.
- The rest are placed on random nodes with restrictions that no more than one replica is placed at any one node and no more than two replicas are placed in the same rack, if possible.

- The choice to place the second and third replicas on a different rack better distributes the block replicas for a single file across the cluster. If the first two replicas were placed on the same rack, for any file, two-thirds of its block replicas would be on the same rack.
- After all target nodes are selected, nodes are organized as a pipeline in the order of their proximity to the first replica. Data are pushed to nodes in this order.
- For reading, the NameNode first checks if the client's host is located in the cluster. If yes, block locations are returned to the client in the order of its closeness to the reader. The block is read from DataNodes in this preference order.

3.3.3 File System Namespace

- HDFS supports a traditional hierarchical file organization in which a user or an application can create directories and store files inside them.
- The file system namespace hierarchy is similar to most other existing file systems; you can create, rename, relocate, and remove files.
- HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service.
- **Data replication :** HDFS replicates file blocks for fault tolerance. An application can specify the number of replicas of a file at the time it is created, and this number can be changed any time after that. The name node makes all decisions concerning block replication.
- HDFS uses an intelligent replica placement model for reliability and performance. Optimizing replica placement makes HDFS unique from most other distributed file systems, and is facilitated by a rack-aware replica placement policy that uses network bandwidth efficiently.
- **Data organization :** One of the main goals of HDFS is to support large files. The size of a typical HDFS block is 64MB. Therefore, each HDFS file consists of one or more 64MB blocks. HDFS tries to place each block on separate data nodes.
- HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed.
- HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients.
- HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories.
- HDFS does not support hard links or soft links. Files in HDFS are write-once and have strictly one writer at any time.

3.3.4 MapReduce

- MapReduce is a programming model and software framework first developed by Google. Intended to facilitate and simplify the processing of vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner.

Characteristics of MapReduce :

1. Very large scale data : peta, exa bytes.
2. Write once and read many data. It allows for parallelism without mutexes.
3. Map and Reduce are the main operations: simple code.
4. All the map should be completed before reduce operation starts.
5. Map and reduce operations are typically performed by the same physical processor.
6. Number of map tasks and reduce tasks are configurable.

Data flow in the MapReduce programming model

- MapReduce : A programming model to facilitate the development and execution of distributed tasks.
- The programmer defines the program logic as two functions :
  a. Map transforms the input into key-value pairs to process.
  b. Reduce aggregates the list of values for each key.
- The MapReduce environment takes in charge distribution aspects. A complex program can be decomposed as a succession of Map and Reduce tasks Higher-level languages (Pig, Hive, etc.) help with writing distributed applications.
- MapReduce is a parallel programming model especially dedicated for complex and distributed computations, which has been derived from the functional paradigm.
- In general, MapReduce processing is composed of two consecutive stages, which for most problems are repeated iteratively: the map and the reduce phase. Fig. 3.3.3 shows data flow in the MapReduce programming model.
- Map processes the data on hosts in parallel, whereas the reduce aggregates the results. At each iteration independently, the whole data is split into chunks, which in turn, are used as the input for mappers.
- Each chunk may be processed by only one mapper. Once the data is processed by mappers, they can emit View the MathML source?key,value? pairs to the reduce phase.

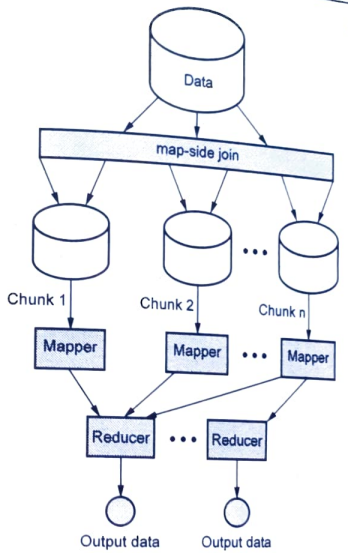

Fig. 3.3.3 Data flow in map reduce

- Before the reduce phase the pairs are sorted and collected according to the View the MathML sourcekey values, therefore each reducer gets the list of values related to a given View the MathML sourcekey. The consolidated output of the reduce phase is saved into the distributed file system.

MapReduce processes big data :

- With HDFS, we are able to distribute the data so that data is stored on hundreds of nodes instead of a single large machine.
- Mapreduce provides the framework for highly parallel processing of data across clusters of commodity hardware. Fig. 3.3.4 shows MapReduce data processing.
- It removes the complicated programming part from the programmers and moves into the framework. Programmers can write simple programs to make use of the parallel processing.
- The framework splits the data into smaller chunks that are processed in parallel on cluster of machines by programs called mappers.

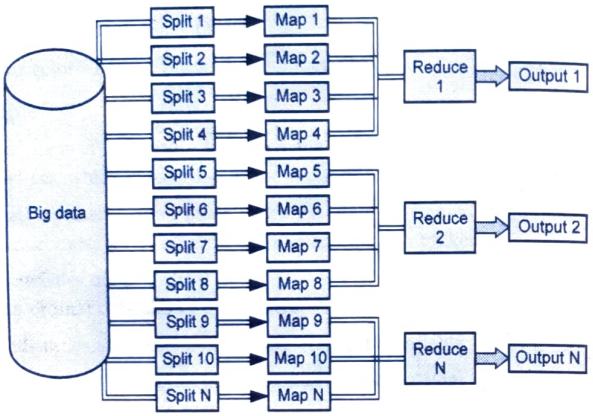

Fig. 3.3.4 MapReduce data processing

- The output from the mappers is then consolidated by reducers into desired result. The share nothing architecture of mappers and reducers make them highly parallel.
- Data locality is achieved by mapreduce by working closely with HDFS. When you specify the file system as HDFS for mapreduce, it automatically schedules the mappers on the same node as where the block of data exists.
- Mapreduce can get the blocks from HDFS and process them. The final output from Mapreduce also can be stored in HDFS file system. However, the intermediate files between mappers and reducers are not stored in HDFS and are stored on the local file system of the mappers.

Hadoop / MapReduce limitations :

1. Cannot control the order in which the maps or reductions are run.
2. For maximum parallelism, you need Maps and Reduces to not depend on data generated in the same MapReduce job (i.e. stateless).
3. A database with an index will always be faster than a MapReduce job on unindexed data.
4. Reduce operations do not take place until all Maps are complete.
5. General assumption that the output of Reduce is smaller than the input to Map large data source used to generate smaller final values.

3.3.4.1 Functions of Job Tracker and Task Tracker

Function of Job tracker :

- There is a single job tracker that runs on the master node. It is the driver for the map - reduce jobs. Its functions are :
  1. Accepts jobs from client and divides into tasks.
  2. Schedules tasks on worker nodes called task trackers.
  3. Keeps heartbeat info from task trackers on worker nodes.
  4. Reschedules the task on alternate worker if a worker fails.

Function of task tracker :

- Task tracker runs on each worker node and there are as many task trackers as the worker nodes. If HDFS is also used, then data nodes of HDFS also become worker nodes for task tracker. The functions of a task tracker are :
  a. Takes assignments from job tracker.
  b. Executes the tasks locally.
  c. Each worker node has specific number of mapper and reducer tasks it can take at one time.
  d. The tasks assigned are run in parallel.
  e. Normally they can take more map jobs than reduce tasks.
  f. Task tracker does a task attempt before executing task.
  g. Task tracker may do multiple attempts before declaring a task as failed.
  h. Task tracker maintains a connection with the task attempt called umbilical protocol.
  i. Task tracker sends a regular heartbeat signal to job tracker indicating its status including available map and reduce tasks.
  j. Task tracker runs each task attempt in a separate JVM. So even if the task has bad code due to which it fails, it will not cause task tracker to abort.

3.3.5 Hadoop Ecosystem

- Hadoop ecosystem is neither a programming language nor a service, it is a platform or framework which solves big data problems.
- The Hadoop ecosystem refers to the various components of the Apache Hadoop software library, as well as to the accessories and tools provided by the Apache Software Foundation for these types of software projects and to the ways that they work together.

- Hadoop is a Java - based framework that is extremely popular for handling and analysing large sets of data. 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.
- Some of the most well - known tools of the Hadoop ecosystem include HDFS, Hive, Pig, YARN, MapReduce, Spark, HBase, Oozie, Sqoop, Zookeeper, etc.
- Fig. 3.3.5 shows Apache Hadoop ecosystem.

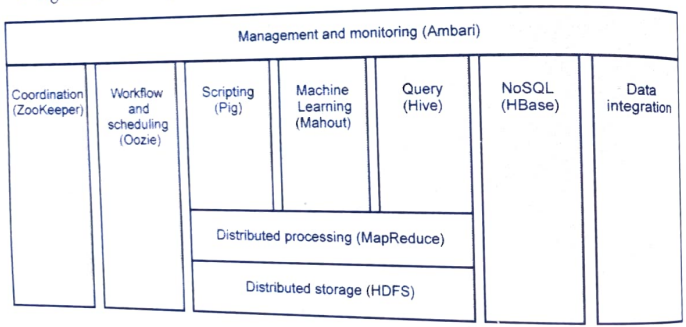

Fig. 3.3.5 : Apache Hadoop ecosystem

- Hadoop Distributed File System (HDFS), is one of the largest Apache projects and primary storage system of Hadoop. It employs a NameNode and DataNode architecture. It is a distributed file system able to store large files running over the cluster of commodity hardware.
- YARN stands for Yet Another Resource Negotiator. It is one of the core components in open source Apache Hadoop suitable for resource management. It is responsible for managing workloads, monitoring and security controls implementation.
- Hive is an ETL and Data warehousing tool used to query or analyze large datasets stored within the Hadoop ecosystem. Hive has three main functions : Data summarization, query and analysis of unstructured and semi - structured data in Hadoop.

- Map - Reduce : It is the core component of processing in a Hadoop Ecosystem as it provides the logic of processing. In other words, MapReduce is a software framework which helps in writing applications that processes large data sets using distributed and parallel algorithms inside Hadoop environment.
- Apache Pig is a high - level scripting language used to execute queries for larger datasets that are used within Hadoop.
- Apache Spark is a fast, in - memory data processing engine suitable for use in a wide range of circumstances. Spark can be deployed in several ways, it features Java, Python, Scala and R programming languages and supports SQL, streaming data, machine learning and graph processing, which can be used together in an application.
- Apache HBase is a Hadoop ecosystem component which is a distributed database that was designed to store structured data in tables that could have billions of rows and millions of columns. HBase is scalable, distributed and NoSQL database that is built on top of HDFS. HBase provide real - time access to read or write data in HDFS.

3.3.6 Pig

- Pig is an open - source high level data flow system. A high - level platform for creating MapReduce programs used in Hadoop. It translates into efficient sequences of one or more MapReduce jobs.
- Pig offers a high - level language to write data analysis programs which we call as Pig Latin. The salient property of pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.
- Pig makes use of both, the Hadoop Distributed File System as well as the MapReduce.
- Features of Pig Hadoop :
  1. In-built operators : Apache Pig provides a very good set of operators for performing several data operations like sort, join, filter, etc.
  2. Ease of programming.
  3. Automatic optimization : The tasks in Apache Pig are automatically optimized.
  4. Handles all kinds of data : Apache Pig can analyze both structured and unstructured data and store the results in HDFS.

- Fig. 3.3.6 shows Pig architecture.

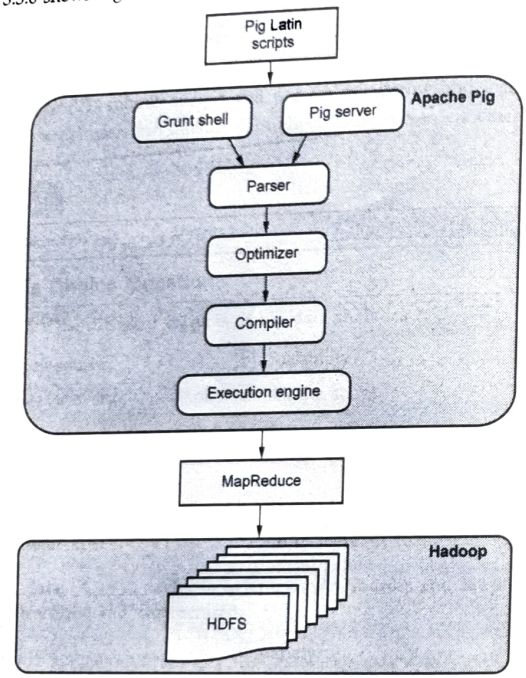

Fig. 3.3.6 Pig architecture

- Pig has two execution modes :
  1. **Local mode :** To run pig in local mode, we need access to a single machine; all files are installed and run using our local host and file system. Specify local mode using the -x flag (pig-x local).
  2. **Mapreduce mode** : To run pig in mapreduce mode, we need access to a Hadoop cluster and HDFS installation. Mapreduce mode is the default mode; we can, but don't need to, specify it using the -x flag.
- Pig Hadoop framework has four main components :
  1. **Parser :** When a Pig Latin script is sent to Hadoop Pig, it is first handled by the parser. The parser is responsible for checking the syntax of the script, along with other miscellaneous checks. Parser gives an output in the form of a Directed Acyclic Graph (DAG) that contains Pig Latin statements, together with other logical operators represented as nodes.
  2. **Optimizer :** After the output from the parser is retrieved, a logical plan for DAG is passed to a logical optimizer. The optimizer is responsible for carrying out the logical optimizations.
  3. **Compiler** : The role of the compiler comes in when the output from the optimizer is received. The compiler compiles the logical plan sent by the optimize The logical plan is then converted into a series of MapReduce tasks or jobs.
  4. **Execution engine** : After the logical plan is converted to MapReduce jobs, these jobs are sent to Hadoop in a properly sorted order and these jobs are executed on Hadoop for yielding the desired result.
- Pig can run on two types of environments : The local environment in a single JVM or the distributed environment on a Hadoop cluster.
- Pig has variety of scalar data types and standard data processing options. Pig supports Map data; a map being a set of key - value pairs.
- Most pig operators take a relation as an input and give a relation as the output. It allows normal arithmetic operations and relational operations too.
- Pig's language layer currently consists of a textual language called Pig Latin. Pig Latin is a data flow language. This means it allows users to describe how data from one or more inputs should be read, processed and then stored to one or more outputs in parallel.
- These data flows can be simple linear flows or complex workflows that include points where multiple inputs are joined and where data is split into multiple streams to be processed by different operators. To be mathematically precise, a Pig Latin script describes a Directed Acyclic Graph (DAG), where the edges are data flows and the nodes are operators that process the data.
- The first step in a Pig program is to LOAD the data user want to manipulate from HDFS. Then user run the data through a set of transformations. Finally, user DUMP the data to the screen or user STORE the results in a file somewhere.

Advantages of Pig :

1) Fast execution that works with MapReduce, Spark and Tez.
2) Its ability to process almost any amount of data, regardless of size.
3) A strong documentation process that helps new users learn Pig Latin.
4) Local and remote interoperability that lets professionals work from anywhere with a reliable connection.

Pig disadvantages :

1) Slow start - up and clean - up of MapReduce jobs.
2) Not suitable for interactive OLAP Analytics.
3) Complex applications may require many user defined function.

3.3.7 Hive

- Apache Hive is an open source data warehouse software for reading, writing and managing large data set files that are stored directly in either the Apache Hadoop Distributed File System (HDFS) or other data storage systems such as Apache HBase.
- Data analysts often use Hive to analyze data, Query large amounts of unstructured data and generate data summaries.
- Features of Hive :
  1. It stores schema in a database and processed data into HDFS.
  2. It is designed for OLAP.
  3. It provides SQL type language for querying called HiveQL or HQL.
  4. It is familiar, fast, scalable and extensible.
- Hive supports variety of storage formats : TEXTFILE for plaintext, SEQUENCEFILE for binary key - value pairs, RCFILE stores columns of a table in a record columnar format.
- Hive table structure consists of rows and columns. The rows typically correspond to some record, transaction or particular entity detail.
- The values of the corresponding columns represent the various attributes or characteristics for each row.
- Hadoop and its ecosystem are used to apply some structure to unstructured data. Therefore, if a table structure is an appropriate way to view the restructured data, Hive may be a good tool to use.
- Following are some Hive use cases :
  1. Exploratory or ad-hoc analysis of HDFS data : Data can be queried, transformed and exported to analytical tools.
  2. Extracts or data feeds to reporting systems, dashboards or data repositories such as HBase.
  3. Combining external structured data to data already residing in HDFS.

Advantages :

1) Simple querying for anyone already familiar with SQL.
2) Its ability to connect with a variety of relational databases, including Postgres and MySQL.
3) Simplifies working with large amounts of data.

Disadvantages :

1) Updating data is complicated.
2) No real time access to data.
3) High latency.

3.3.8 Difference between Pig and Hive

Sr. No.	Pig	Hive
1.	Pig used for data transformations and processing.	Hive used for warehousing and querying data.
2.	Pig works on structured, semi - structured and unstructured data.	Hive works only on structured data.
3.	Pig does not support web interface.	Hive support web interface.
4.	Pig is a scripting platform that runs on Hadoop clusters, designed to process and analyze large datasets. Pig uses a language called Pig Latin, which is similar to SQL.	Hive is a data warehouse system used to query and analyze large datasets stored in HDFS. Hive uses a query language called HiveQL, which is similar to SQL.
5.	Pig support Avro file format	Hive does not support Avro file format
6.	Creating schema is not required to store data in Pig	Hive supports schema
7.	Pig loads data quickly	Hive takes time to load but executes quickly
8.	Pig works on the client - side of the cluster	Hive works on the server - side of the cluster
9.	Used for programming.	Used for reporting

3.3.9 HBase

- HBase is an open source, non - relational, distributed database modeled after Google's BigTable. HBase is an open source and sorted map data built on Hadoop. It is column oriented and horizontally scalable.

- It is a part of the Hadoop ecosystem that provides random real - time read/write access to data in the Hadoop file system. It runs on top of Hadoop and HDFS, providing Big Table - like capabilities for Hadoop.
- HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- HBase supports an easy - to - use Java API for programmatic access. It also supports Thrift and REST for non - Java front - ends.
- HBase is a column oriented distributed database in Hadoop environment. It can store massive amounts of data from terabytes to petabytes. HBase is scalable, distributed big data storage on top of the Hadoop eco system.
- The HBase physical architecture consists of servers in a Master-Slave relationship. Typically, the HBase cluster has one Master node, called HMaster and multiple Region Servers called HRegionServer.
- Fig. 3.3.7 shows Hbase architecture.

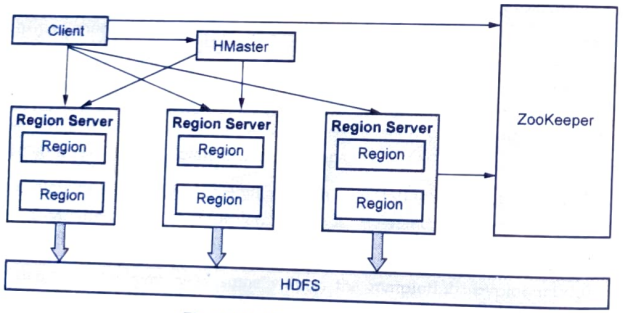

Fig. 3.3.7 Hbase architecture

- Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. If the client wants to communicate with regions servers, client has to approach Zookeeper.
- HMaster in the master server of Hbase and it coordinates the HBase cluster. HMaster is responsible for the administrative operations of the cluster.
- HRegions servers : It will perform the following functions in communication with HMaster and Zookeeper.
  1. Hosting and managing regions.
  2. Splitting regions automatically.

  3. Handling read and writes requests.
  4. Communicating with clients directly.
- **HRegions :** For each column family, HRegions maintain a store. Main components of HRegions are Memstore and Hfile
- Data model in HBase is designed to accommodate semi - structured data that could vary in field size, data type and columns.
- HBase is a column - oriented, non - relational database. This means that data is stored in individual columns, and indexed by a unique row key. This architecture allows for rapid retrieval of individual rows and columns and efficient scans over individual columns within a table.
- Both data and requests are distributed across all servers in an HBase cluster, allowing you to query results on petabytes of data within milliseconds. HBase is most effectively used to store non - relational data, accessed via the HBase API.

3.3.10 Difference between HDFS and HBase

Sr. No.	HDFS	HBase
1.	HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
2.	HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
3.	It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
4.	It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.
5.	HDFS are suited for high latency operations.	Hbase is suited for low latency operations.
6.	In HDFS, data are primarily accessed through Map Reduce jobs.	Hbase provides access to single rows from billions of records.
7.	HDFS doesn't have the concept of random read and write operations.	Hbase data is accessed through shell commands, client API in Java, REST, Avro or Thrift.

3.3.11 Mahout

- Mahout is an open source machine learning library from Apache written in java. It also supports a number of clustering algorithms like k - means, mean - shift and canopy.
- The primitive features of Apache Mahout include :
  1. The algorithms of Mahout are written on top of Hadoop, so it works well in distributed environment.
  2. Mahout uses the Apache Hadoop library to scale effectively in the cloud.
  3. Mahout offers the coder a ready - to - use framework for doing data mining tasks on large volumes of data.
  4. Mahout lets applications to analyze large sets of data effectively and in quick time.
  5. Includes several MapReduce enabled clustering implementations such as k - means, fuzzy k - means, Canopy etc.
  6. Supports Distributed Naive Bayes and Complementary Naïve Bayes classification implementations.
  7. Comes with distributed fitness function capabilities for evolutionary programming.
  8. Includes matrix and vector libraries.
- Mahout is an open source machine learning library built on top of Hadoop to provide distributed analytics capabilities. Mahout incorporates a wide range of data mining techniques including collaborative filtering, classification and clustering algorithms.

3.3.12 Hadoop Configuration File

- Hadoop configs are contained under /etc/hadoop/conf in CDH.

Sr. No.	Name of File	Description
1.	hadoop - env.sh	• Used for environment - specific settings • It update the JAVA path to configure user JAVA_HOME. • It also specify JVM options for various Hadoop components.
2.	core - site.xml	• System - level Hadoop configuration items, such as the HDFS URL, • It configure the Hadoop temporary directory and script locations for rack - aware Hadoop clusters.
3.	hdfs - site.xml	• Used for HDFS settings such as file replication count, the block size, permissions.

4.	mapred - site.xml	• Hadoop distributed file settings i.e. no. of reduce tasks, memory sizes
5.	Masters	• List of hosts that are Hadoop masters, i.e. secondary name nodes
6.	slaves	• List of set of hosts that are going to act as slaves

- The default settings for above configuration are available at **http://hadoop.apache.org/common/docs/r1.0.0/core-default.html**

3.3.13 Hadoop Distributed File System

- Hadoop Distributed File System (HDFS) is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds of nodes.
- A Block is the minimum amount of data that it can read or write. HDFS blocks are 128 MB by default and this is configurable. When a file is saved in HDFS, the file is broken into smaller chunks or "blocks".
- HDFS is a fault - tolerant and resilient system, meaning it prevents a failure in a node from affecting the overall system's health and allows for recovery from failure too. In order to achieve this, data stored in HDFS is automatically replicated across different nodes.
- HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another or rename a file.
- Hadoop Distributed File System is a block - structured file system where each file is divided into blocks of a pre - determined size. These blocks are stored across a cluster of one or several machines.
- Apache Hadoop HDFS Architecture follows a Master/Slave Architecture, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).
- HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.
- HDFS read write operation is as follows :
- HDFS uses a single - write, multiple - read model, where the files are written once and read several times.

- The data cannot be altered once written. However, data can be appended to the file by reopening it. All files in the HDFS are saved as data blocks.
- To write a in HDFS, a client needs to interact with master i.e. namemode (master). Now namenode provides the address of the datanotes (slaves) on which client start writing the data. Client directly writes data on the datanodes, now datanode will create data write pipeline.
- The first datanote will copy the block to another datanode, which intern copy it to the third datanode. Once it creates the replicas of blocks, it sends back the acknowledgment.

3.3.14 Hadoop Shell Commands

- Simple shell commands can be written for Hadoop application by considering HDFS structure. Following are some of the frequent shell commands which can be used during hadoop operations.

1) Upload and download a file in HDFS

Upload :

hadoop fs - put

Copy single src file or multiple src files from local file system to the Hadoop data file system.

Syntax :

```
hadoopfs-put <localsrc> ... < HDFS_dest_Path>
Example :
hadoop fs_put/home/DDL/Samplefile.txt/user/user/DDL/dir3
```

Download

hadoop fs - get :

Copies /Downloads files to the local file system

syntax :

```
hadoop fs-get <hdfs_src> <localdst>
Example :
hadoop fs-get/user/DDL/dir3/Samplefile.txt/home/
```

2) See contents of a file

Same as unix cat command :

Syntax :

```
hadoop fs-cat <path(filename)>
```


Example :

```
hadoop fs-cat/user/DDL/dir1/abc.txt
```

3) Copy a file from source to destination

This command allow multiple sources as well in which case the destination must be a directory.

Syntax :

```
hadoop fs-cp <source> <dest>
Example :
hadoop fs - cp/user/DDL/dir1/abc.txt/user /user/DDL/dir2
```

Copy a file from/To Local file system to HDFS

copyFromLocal

Syntax :

```
hadoop fs-copy FromLocal<localsrc> URI
Example :
hadoop fs-coopyFromLocal/home/DDL/pqr.txt/user/DDL/pqr.txt
```

Similar to put command, except that the source is restricted to a local file reference.

copyToLocal

Syntax :

```
hadoop fs-copyTolocal[-ignorecrc][-crc] URI<localdst>
```

Similar to get command, except that the destination is restricted to a local file reference.

4) Move file from source to destination.

Note : Moving file across filesystem is not permitted.

Syntax :

```
hadoop fs-mv <src> <dest>
Example :
hadoop fs-mv/user/DDL/dir1/abc.txt/user/DDL/dir2
```

5) Remove a file or directory in HDFS

Remove files specified as argument. Deletes directory only when it is empty.

Syntax :

```
hadoop fs-rm <arg>
Example :
hadoop fs-rm /user/DDL/dir1/abc.txt
```

Recursive version of delete.

Syntax :

```
hadoop fs-rmr<arg>
```

Example :
hadoop fs-rmr/user/DDL/

3.3.15 Hadoop Job Execution

- Fig 3.3.8 shows job execution sequence of Hadoop.

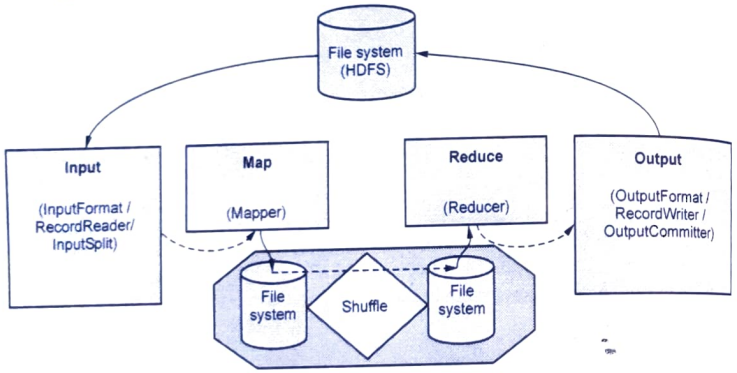

Fig. 3.3.8 Job execution sequence of Hadoop

- Daemon service in Hadoop environment is job Tracker, which is responsible for job execution and processing. For every hadoop cluster there is only one Job Tracker, run on JVM process. Its responsible for executing all job requests made by client application. The process of execution of job sequence is as follows :
- Job request is submitted by client.
- Job Tracker processes the job request
  - Job Trackers communicates the NameNode for getting the data location in the cluster.
  - Job Tracker finds Task Tracker modes in DataNodes and issues job to these nodes.
  - Job Tracker notifies Job Tracker about the status of Job completion.
- Upon completion of Job, the Job Tracker update the status and client is signaled about the completion of processing.
- Task Tracker node accepts jobs from Job Tracker in cluster. Task Tracker maintains set of slots for maintaining the details of total number of tasks it can accept at given point of time. Following are the responsibilities of Task Tracker :
- Task Tracker creates separate JVM process for task, so the particular task can be isolated and failure of that does not affect the Task Tracker.

- Task Tracker monitors job execution created by it and maintains the output and exit codes. It notifies Job Tracker the status of tasks.
- The Task Tracker communicates heartbeats to job Tracker, which indicates availability of node and informs number of available slots for further job allocation by Job Tracker.
- Upon failure of Task Tracker, Job Tracker can resubmit job to other nodes or it can mark part of record to avoid processing of portion of data or can blacklist Task Tracker.
- The Job Tracker and Task Tracker perform job management in Hadoop through MapReduce processes. HDFS also responsible for load balancing block allocation, disk management etc.

Review Questions

1. *Explain Hadoop ecosystem in detail.* **SPPU : April-18 (In Sem), Marks 6**
2. *Explain HDFS read and write operation in detail.* **SPPU : April-18, 19 (In Sem), Marks 6**
3. *What is the need of map reduce in big data ? Define the architecture of Map-Reduce on Hadoop.* **SPPU : April-18 (In Sem), Marks 4**
4. *Explain job execution in Hadoop with example.* **SPPU : May-18 (End Sem), Marks 6**
5. *What are the advantages of Hadoop ? Explain Hadoop architecture and its components with proper diagram.* **SPPU : Dec.-18 (End Sem), Marks 5**
6. *Explain the concept of blocks and heartbeat mechanism in HDFS architecture.* **SPPU : Dec.-18 (End Sem), Marks 5**
7. *List and explain any five hadoop shell commands with syntax.* **SPPU : Dec.-18 (End Sem), Marks 5**
8. *Explain term heartbeat mechanism in HDFS.* **SPPU : April-19 (In Sem), Marks 3**
9. *Explain HDFS read and write operations in detail.* **SPPU : April-19 (In Sem), Marks 6**
10. *What is the role of sorter, shuffler and combiner in map reduces paradigm ?* **SPPU : March-19 (In Sem), Marks 4**
11. *Write two Hadoop shell file management command.* **SPPU : Dec.-19 (End Sem), Marks 4**
12. *Explain map reduce with proper diagram for word count example.* **SPPU : Dec.-19 (End Sem), Marks 4**
13. *Explain anatomy of file read and write in HDFS.* **SPPU : Dec.-19 (End Sem), Marks 6**
14. *Explain the role of job and task tracker in the execution and processing of job within Hadoop environment, with architecture.* **SPPU : April-20 (In Sem), Marks 4**
15. *Draw and explain map reduce architecture.* **SPPU : April-20 (In Sem), Marks 6**

3.4 Introduction to NOSQL

SPPU : April-18, 19, 20, Dec.-18

- NoSQL means Not Only SQL, it solves the problem of handling huge volume of data that relational databases cannot handle. NoSQL databases are schema free and are non - relational databases. Most of the NoSQL databases are open source.

Why NoSQL ?

- It can handle large volumes of structured, semi-structured and unstructured data.
- Agile sprints, quick iteration and frequent code pushes.
- Object - oriented programming that is easy to use and flexible.
- Scale - out architecture,

Types of NoSQL Stores -

1. Column oriented (Accumlo, Cassandra, HBase)
2. Document Oriented (MpngoDB, Couchbase, Clusterpoint)
3. Key - value (Dynamo, MemcacheDB, Riak)
4. Graph (Allegro, Neo4j, OrientDB)_

- Features of NoSQL

1) Multi-model : The concept is to allow multiple data models in a single database.
2) Distributed : NoSQL databases use the shared - nothing architecture, implying that the database has no single control unit or storage.
3) Eliminated downtime : The data is maintained at various nodes owing to its architecture, the failure of one node will not affect the entire system.
4) NoSQL databases can process structured, semi-structured or unstructured data with the same ease, thereby increasing performance.
5) High Scalability : NoSQL databases use horizontal scaling and thus the data remains accessible even when one or more nodes go down.

- Types of NoSQL

a) Key - value data stores :

- Key - value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high - speed read and write processing of non - transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key.
- The application has complete control over what is stored in the value, making this the most flexible NoSQL model.

- Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high - velocity, non - transactional data.

b) Document stores :

- Document databases typically store self - describing JSON, XML and BSON documents. They are similar to key - value stores, but in this case, a value is a single document that stores all data related to a specific key.
- Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.

c) Wide - column stores :

- Wide - column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table.
- Wide - column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved.
- In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

d) Graph stores :

- A graph database uses graph structures to store, map and query relationships. They provide index - free adjacency, so that adjacent elements are linked together without using an index.

3.4.1 CAP Theorem

- The three parts of the CAP Theorem are Consistency, Availability, and Partition Tolerance.
- The CAP Theorem is a fundamental theorem in distributed systems that states any distributed system can have at most two of the following three properties.
  a) Consistency
  b) Availability
  c) Partition tolerance

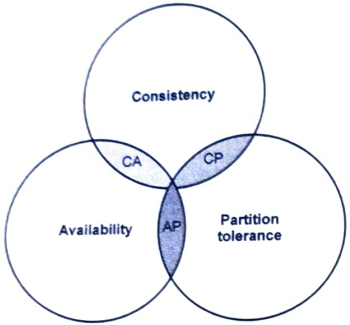

Fig. 3.4.1 CAP theorem

a) Consistency : All reads receive the most recent write or an error.

b) Availability : All reads contain data, but it might not be the most recent.

c) Partition tolerance : The system continues to operate despite network failures (i.e; dropped partitions, slow network connections or unavailable network connections between nodes.)

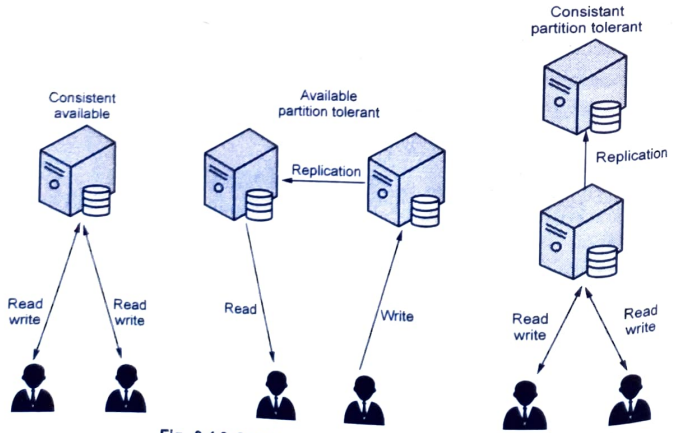

Fig. 3.4.2 CAP theorem read-write operation

- When CAP Theorem was proposed, the understanding was that system designers had three options :
  a) **CA systems :** Sacrifice partition tolerance. Single - site or cluster databases using two-phase commit are examples.
  b) **CP systems :** Sacrifice availability. If there's a partition, for consistency we make the service unavailable : Return a timeout error or lock operations.
  c) **AP systems :** Sacrifice consistency. If there's a partition, we continue accepting requests but reconcile them later (writes) or return stale values (reads).
- In NoSQL distributed databases, CAP Theorem has led to the belief that eventual consistency provides better availability than strong consistency. Some believe this is an outdated notion. It's better to factor in sensitivity to network delays.

3.4.2 Difference between SQL and NoSQL

Sr. No.	SQL	NoSQL
1.	These are called RDBMS	These are called not only SQL database.
2.	Based on ACID properties i.e. Atomicity, Consistency, Isolation and Durability.	Based on CAP properties i.e. (Consistency, Availability and Partition tolerance)
3.	These are table based database i.e. the data are stored in a table with rows and columns.	These databases are documented based, key - value pairs or graph based etc.
4.	These are scaled vertically. Load can be managed by increasing CPU, RAM etc in the same server.	These are scaled horizontally. A few servers can be added to manage large traffic.
5.	Preferred for complex, query execution.	Not preferred for complex query execution.
6.	Examples : DB2, MySQL, Oracle, Postgress, SQL server.	Examples : ConchDb, MongoDB, RavenDb, Redis, Cassandra, Hbase, Neo-4j, BigTable.

3.4.3 Textual ETL Processing

- The components of textual ETL (Extract, Transform, Load) processing are Textual ETL rules engine. User Interface, Taxonomies, output database. Fig. 3.4.3 shows ETL processing.

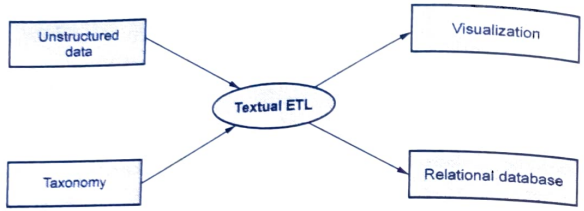

Fig. 3.4.3

- Textual ETL rules Engine takes large unstructured data and parse it to extract value for integration. Rules engines contains series of data processing steps and algorithms such as classification, clustering affinity, proximity.
- Taxonomy integration, master data integration, metadata integration are some integration processing techniques are available in rules engine.
- User Interface helps business users to create and supply processing rules through drag drop and free - form text interface in different languages.
- Taxonomies are required that of several categories of multi - structures and multi - hierarchical data. Third party taxonomy libraries are available, where textual ETL supports variety taxonomies in 16 different languages.
- Output Database in textual ETL is any RDBMS or NoSQL database. To integrate structured and unstructured database uses result sets which are key value pairs.

Review Questions

1. *Differentiate between SQL and NoSQL databases with example. What is the need to develop big data applications using NoSQL databases ?* **SPPU : April-18, 19 (In Sem), Marks 4**
2. *What is textual ETL processing ? Explain different components of textual ETL processing.* **SPPU : Dec.-18 (End Sem), Marks 5**
3. *Write a short note on textual ETL processing.* **SPPU : April-20 (In Sem), Marks 4**

3.5 Multiple Choice Questions

Q.1 **Mahout is an open - source machine learning library from Apache written in _____.**

a C b C++
c Python d Java

Q.2 **HBase is a _____ , non - relational database.**

a row - oriented b column - oriented
c horizontal d vertical

Q.3 **Pig support _____ file format.**

a mp3 b jpeg
c doc d Avro

Q.4 **Pig works on the _____ of the cluster.**

a server - side b master node
c client - side d none

Q.5 **Hive works on the _____ of the cluster.**

a server - side b master node
c client - side d none

Q.6 **MapReduce is a programming model and software framework first developed by _____.**

a Microsoft b Amazon
c TCS d Google

Q.7 **HDFS blocks are _____ MB by default and this is configurable.**

a 32 b 64
c 128 d 256

Q.8 **Apache Hadoop HDFS architecture follows a _____ architecture.**

a client/server b master/slave
c peer to peer d all of these

Q.9 **Hive is _____ and data warehousing tool used to query or analyze large datasets stored within the Hadoop ecosystem.**

a STL b HDFS
c ETL d data mining

Q.10 Hadoop ecosystem include _____.

a Hive
b Pig
c YARN
d all of these

Q.11 An input to a MapReduce is been divided into the fixed size of pieces named as the _____.

a reducer
b input splits
c output splits
d data splits

Answer Keys for Multiple Choice Questions :

Q.1	d	Q.2	b
Q.3	d	Q.4	c
Q.5	a	Q.6	d
Q.7	c	Q.8	b
Q.9	c	Q.10	d
Q.11	b		