

BDA.2

Q.8 Write down the characteristics of big data applications.

Ans. Characteristics of Big Data Applications :

- Data throttling
- Computation- restricted throttling
- Large data volumes
- Significant data variety
- Benefits from data parallelization

Q.9 Write down the four computing resources of big data storage.

Ans. Computing Resources of Big Data Storage :

- Processing Capability
- Memory
- Storage
- Network

Q.10 What are the three modes in which Hadoop can run?

Ans. The three modes in which Hadoop can run are :

- Standalone Mode:** This is the default mode. It uses the local file system and a single Java process to run the Hadoop services.
- Pseudo-distributed Mode:** This uses a single-node Hadoop deployment to execute all Hadoop services.
- Fully-distributed Mode:** This uses separate nodes to run Hadoop master and slave services.

Q.11 How can you restart NameNode and all the daemons in Hadoop?

Ans. The following commands will help you restart NameNode and all the daemons:

You can stop the NameNode with `/sbin /Hadoop-daemon.sh stop NameNode` command and then start the command `/sbin/Hadoop-daemon.sh start NameNode`

You can stop all the daemons with `/sbin /stop-all.sh` command and then start the daemons using the `/sbin/start-all.sh` command.

IT Tech. (VIII Sem.) C.S. Solved Papers

Q.12 Define the Port Numbers for NameNode, Task Tracker and Job Tracker.

Ans. NameNode — Port 50070
Task Tracker — Port 50060
Job Tracker — Port 50030

PART-B

Q.13 Explain the classification of data.

Ans. Classification of Data : Data can be classified as structured, semi-structured, multi-structured and unstructured.

Structured data conform and associate with data schemas and data models. Structured data are found in tables (rows and columns). Nearly 15-20% data are in structured or semi-structured form. Unstructured data do not conform and associate with any data models.

Applications produce continuously increasing volumes of both unstructured and structured data. Data sources generate data in three forms, viz. structured, semi-structured and unstructured.

Using Structured Data : Structured data enables the following:

- Data insert, delete, update and append
- Indexing to enable faster data retrieval
- Scalability which enables increasing or decreasing capacities and data processing operations such as storing, processing and analytics
- Transactions processing which follows ACID rules (Atomicity, Consistency, Isolation and Durability)
- Encryption and decryption for data security.

Using Semi-structured Data : Examples of semi-structured data are XML and JSON documents. Semi-structured data contain tags or other markers, which separate semantic elements and enforce hierarchies of records and fields within the data. Semi-structured form of data does not conform and associate with formal data model structures. Data do not associate data models, such as the relational database and table models.

Big Data Analytics

Using Multi-structured Data : Multi-structured data refers to data consisting of multiple formats of data, viz. structured, semi-structured and/or unstructured data. Multi-structured data sets can have many formats. They are found in non-transactional systems. For example, streaming data on customer interactions, data of multiple sensors, data at web or enterprise server or the data-warehouse data in multiple formats.

Large-scale interconnected systems are thus required to aggregate the data and use the widely distributed resources efficiently.

Multi- or semi-structured data has some semantic meanings and data is in both structured and unstructured formats. But as structured data, semi-structured data nowadays represent a few parts of data (5-10%). Semi-structured data type has a greater presence compared to structured data.

Using Unstructured Data : Unstructured data does not possess data features such as a table or a database. Unstructured data are found in file types such as TXT, CSV. Data may be as key-value pairs, such as hash key-value pairs. Data may have internal structures, such as in e-mails. The data do not reveal relationships, hierarchy relationships or object-oriented features, such as extensibility. The relationships, schema and features need to be separately established. Growth in data today can be characterised as mostly unstructured data.

Q.14 What are the characteristics of big data?

Ans. Characteristics of big data, called 3Vs (and 4Vs also used) are:

- Volume :** The phrase 'Big Data' contains the term big, which is related to size of the data and hence the characteristic. Size defines the amount or quantity of data, which is generated from applications. The size determines the processing considerations needed for handling that data.
- Velocity :** The term velocity refers to the speed of generation of data. Velocity is a measure of how fast the data generates and processes. To meet the demands and the challenges of processing big data, the velocity of generation of data plays a crucial role.
- Variety :** Big data comprises of a variety of data. Data is generated from multiple sources in a system. This introduces variety in data and therefore introduces 'complexity'. Data consists of various forms and formats. The variety is due to the availability of a large number of

BDA.3

heterogeneous platforms in the industry. This means that the type to which big data belongs to is also an important characteristic that needs to be known for proper processing of data. This characteristic helps in effective use of data according to their formats, thus maintaining the importance of big data.

(iv) **Veracity :** It is also considered an important characteristic to take into account the quality of data captured, which can vary greatly, affecting its accurate analysis.

The 4Vs (i.e. volume, velocity, variety and veracity) data need tools for mining, discovering patterns, business intelligence, artificial intelligence (AI), machine learning (ML), text analytics, descriptive and predictive analytics, and the data visualization tools.

Q.15 How does such a toy company optimize the services offered, products and schedules, devise ways and use Big Data processing and storing for predictions using analytics?

Ans. Assume that a retail and marketing company of toys uses several Big Data sources, such as (i) machine-generated data from sensors (RFID readers) at the toy packaging, (ii) transactions data of the sales stored as web data for automated reordering by the retail stores and (iii) tweets, Facebook posts, e-mails, messages, and web data for messages and reports.

The company uses Big Data for understanding the toys and themes in present days that are popularly demanded by children, predicting the future types and demands. The company using such predictive analytics, optimizes the product mix and manufacturing processes of toys. The company optimizes the services to retailers by maintaining toy supply schedules. The company sends messages to retailers and children using social media on the arrival of new and popular toys.

Q.16 Give an example of features of 3Vs in Big Data and application.

Ans. Consider satellite images of the Earth's atmosphere and its regions. The volume of data from the satellites is large. A number of Indian satellites, such as KALPAHA, INSAT-1A and INSAT-3D generate this data. Foreign satellites also generate voluminous data continuously. Satellites record the images of full disk and sectors, such as east and west Asia sectors and regions.

manageable size. Graphs and charts are some of compressed data.

Data require processing or refinement and they are capable of leading to conclusions. Conclusions can be drawn only after processing or refinement.

detail about storage considerations in

comment intended to support the analysis of data, there must be the infrastructure of a lifecycle from acquisition, preparation, execution. The need to acquire and manage data suggests a need for specialty storage modules for the big data applications. When by storage offerings, some variables to

which looks at whether expectations for improvement are aligned with the storage resources, and the degree to which the system can support massive data increasing size.

which examines how flexible the storage architecture is in allowing the system to be at the constraint of artificial limits.

ty, which looks at any limitations or in providing simultaneous access to an user community without compromising

ce, which involves the storage environment ability to recover from intermittent failures.

IO capacity, which measures whether the channels can satisfy the demanding tuning for absorbing, storing, and sharing large

which measures how well the storage can be integrated into the production

age framework involves a software layer of storage resources and providing utilities. The software configures storage provide a level of fault tolerance, as well as operations using standard protocols (such as

UDP or TCP/IP) among the different processing nodes. In addition, some frameworks will replicate stored data, providing redundancy in the event of a fault or failure.

Q.20 What are the various sources of Big Data?

Ans. Sources of Big Data : Here various sources of big data are briefed. Digitization of content by industries is the new source of data (Villars et al., 2011). Advancements in technology also lead to high rate of data generation. For example, one of the biggest surveys in Astronomy, Sloan Digital Sky Survey (SDSS) has recorded a total of 25TB data during their first (2000-2005) and second surveys (2005-2008) combined. With the advancements in the resolution of the telescope, the amount of data collected at the end of their third survey (2008-14) is 100 TB. Use of "smart" instrumentation is another source of big data. Smart meters in the energy sector record the electricity utilization measurement every 15 minutes as compared to monthly readings before.

In addition to social media, Internet of Things (IoT) has, now, become the new source of data. The data can be captured from agriculture, industry, medical care, etc of the smart cities developed based on IoT. Table summarizes the various types of data produced in different sectors.

Table : Different Sources of Data

Sector	Data Produced	Use
Astronomy	Movement of stars, satellites, etc.	To monitor the activities of asteroid bodies and satellites
Financial	News content via video, audio, twitter and news report	To make trading decisions
Healthcare	Electronic medical records and images	To aid in short-term public health monitoring and long-term epidemiological research programs
Internet of Things (IoT)	Sensor data	To monitor various activities in smart cities

BDA.5

Life Sciences	Gene sequences	To analyze genetic variations and potential treatment effectiveness
Media/Entertainment	Content and user viewing behavior	To capture more viewers
Social Media	Blog posts, tweets, social networking sites, log details	To analyze the customer behavior pattern
Telecommunications	Call Detail Records (CDR)	Customer churn management
Transportation, Logistics, Retail, Utilities	Sensor data generated from fleet transceivers, RFID tag readers and smart meters	To optimize operations
Video Surveillance	Recordings from CCTV to IPTV cameras and recording system	To analyze behavioral patterns for service enhancement and security

Q.21 Write short note on Hadoop architecture.

Ans. Hadoop framework includes following four modules:

- (i) **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
- (ii) **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
- (iii) **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.
- (iv) **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.

We can use following diagram to depict these four components available in Hadoop framework.

Since the base mo of additional of or alongs Apache HB

Q.22 Expla

Ans. The p manage TaskTracke resource ava (track the p

Some

- (i) It is a Data
- (ii) It co locati
- (iii) It tra
- (iv) It allo slots.
- (v) It mo job re
- (vi) It fin tasks

Q.23 Can so, ha

the same as for interval scales.

respond to "I enjoy
pany email" with a
ie characteristics of
ses or observations

- **Appropriate Statistics for Ratio Scales:** Count, frequencies, mode, median, mean, standard deviation (and variance), skewness, and kurtosis.
- **Displays:** Histograms or bar charts, line charts, and scatter plots.

distance. That is,
the same as 4 to 5.
e and two is half of
metic operations on

with interval scales.
l in a manner that
measure from 1 to 9
ause we added 10 to
val scale is the same
ed 5 from all values.
zero, zero remains
appears in the scale

rval Scales: Count,
l, standard deviation
rtosis.

arts, line charts, and

r as nominal scales

eristics:

ses or observations

ve an interpretable

).

a ratio scale:

all prefer \$100 to \$1!

; no income (or, in
e exactly equals our

t \$20 appears as twice

Q.29 Explain in detail about HDFS.

Ans. Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model.

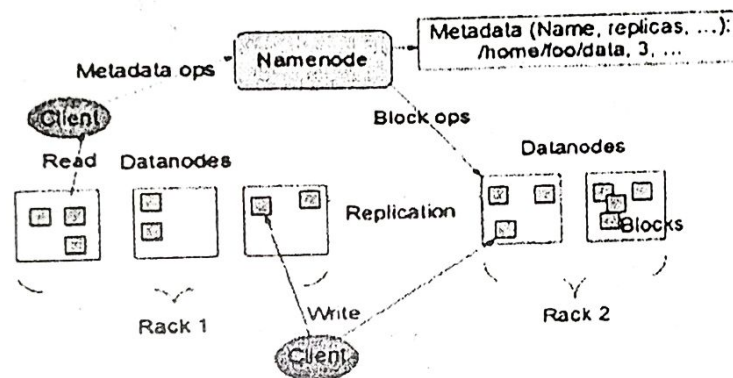


Fig. 1

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 (and even thousands) of nodes. HDFS is one of the major components of Apache Hadoop, the others being MapReduce and YARN. HDFS should not be confused with or replaced by Apache HBase, which is a column-oriented non-relational database management system that sits on top of HDFS and can better support real-time data needs with its in-memory processing engine.

Fast Recovery from Hardware Failures : Because one HDFS instance may consist of thousands of servers, failure of at least one server is inevitable. HDFS has been built to detect faults and automatically recover quickly.

Access to Streaming Data : HDFS is intended more for batch processing versus interactive use, so the emphasis in the design is for high data throughput rates, which accommodate streaming access to data sets.

Education : With the advent of computerized course modules, it is possible to assess the academic performance of the students after each module and give immediate feedback on their learning pattern. This helps the teachers to assess their teaching pedagogy and modify based on the students performance and needs. Dropout patterns, students requiring special attention and students who can handle challenging assignments can be predicted (West, 2012). Beck and Mostow (2008) studied the student reading comprehension using intelligent tutor software and observed that reading mistakes reduced considerably when the students re-read an old story instead of a new story.

Other Sectors : With increasing analytics skills among the various organizations, the advantage of big data analytics can be realized in sectors like construction and material sciences (Brown et al., 2011).

Q.31 Write the challenges of big data analytics in detail.

Ans. Data is a very valuable asset in the world today. The economics of data is based on the idea that data value can be extracted through the use of analytics. Though Big data and analytics are still in their initial growth stage, their importance cannot be undervalued. As big data starts to expand and grow, the importance of big data analytics will continue to grow in everyday lives, both personal and business. In addition, the size and volume of data is increasing every single day, making it important to address the manner in which big data is addressed every day.

According to surveys being conducted many companies are opening up to using big data analytics in their daily functioning. With the rising popularity of Big data analytics, it is but obvious that investing in this medium is what is going to secure the future growth of companies and brands.

The key to data value creation is Big Data Analytics and that is why it is important to focus on that aspect of analytics. Many companies use different methods to employ Big Data analytics and there is no magic solution to successfully implementing this. While data is important, more, important is the process through which companies can gain insights with their help. Gaining insights from data is the goal of big data analytics and that is why investing in a system that can deliver those insights is extremely crucial and important. Successful implementation of big data analytics, therefore, requires a combination of skills, people and

processes that can work in perfect synchronization with each other.

Today, companies are developing at a rapid pace and so are advancements in big technologies. This means that brands must be ready to pilot and adopt big data in such a manner that they become an integral aspect of the information management and analytics infrastructure. With amazing potential, big data is today an emerging disruptive force that is poised to become the next big thing in the field of integrated analytics, thereby transforming the manner in which brands and companies perform their duties across stages and economies.

With great potential and opportunities, however, come great challenges and hurdles. This means that companies must be able to solve all the concerned hurdles so that they can unlock the full potential of big data analytics and its concerned fields. When big data analytics challenges are addressed in a proper manner, the success rate of implementing big data solutions automatically increases. As big data makes its way into companies and brands around the world, addressing these challenges is extremely important.

Some of the major challenges that big data analytics program are facing today include the following :

- 1. Uncertainty of Data Management Landscape:** Because big data is continuously expanding, there are new companies and technologies that are being developed every day. A big challenge for companies is to find out which technology works best for them without the introduction of new risks and problems.
- 2. The Big Data Talent Gap:** While Big Data is a growing field, there are very few experts available in this field. This is because Big data is a complex field and people who understand the complexity and intricate nature of this field are far few and between. Another major challenge in the field is the talent gap that exists in the industry.
- 3. Getting Data into the Big Data Platform:** Data is increasing every single day. This means that companies have to tackle a limitless amount of data on a regular basis. The scale and variety of data that is available today can overwhelm any data practitioner and that is why it is important to make data accessibility simple and convenient for brand managers and owners.
- 4. Need for Synchronization across Data Sources:** As data sets become more diverse, there is a need to incorporate them into an analytical platform. If this is ignored, it can create gaps and lead to wrong insights and messages.

Big Data Analytics

5. Getting Important Insights through the Use of Big Data Analytics: It is important that companies gain proper insights from big data analytics and it is important that the correct department has access to this information. A major challenge in big data analytics is bridging this gap in an effective fashion.

Q.32 What are the problems of traditional large scale systems with Big Data?

Ans. Problem—Schema-On-Write: Traditional systems are schema-on-write. Schema-on-write requires the data to be validated when it is written. This means that a lot of work must be done before new data sources can be analyzed. Here is an example: Suppose a company wants to start analyzing a new source of data from unstructured or semi-structured sources. A company will usually spend months (3-6 months) designing schemas and so on to store the data in a data warehouse. That is 3 to 6 months that the company cannot use the data to make business decisions. Then when the data warehouse design is completed 6 months later, often the data has changed again. If you look at data structures from social media, they change on a regular basis. The schema-on-write environment is too slow and rigid to deal with the dynamics of semi-structured and unstructured data environments that are changing over a period of time. The other problem with unstructured data is that traditional systems usually use Large Object Byte (LOB) types to handle unstructured data, which is often very inconvenient and difficult to work with.

Solution—Schema-On-Read: Hadoop systems are schema-on-read, which means any data can be written to the storage system immediately. Data are not validated until they are read. This enables Hadoop systems to load any type of data and begin analyzing it quickly. Hadoop systems have extremely short business latency compared to traditional systems. Traditional systems require schema-on-write, which was designed more than 50 years ago. A lot of companies need real-time processing of data and customer models generated in hours or days versus weeks or months. The Internet of Things (IoT) is accelerating the data streams coming from different types of devices and physical objects, and digital personalization is accelerating the need to be able to make real-time decisions. Schema-on-read gives Hadoop a tremendous advantage over traditional systems in an area that matters most, that of being able to analyze the data faster to make business decisions. When working with complex data structures that are semi-structured or unstructured, schema-

on-read enables a schema-on-write system.

Problem—Cost of storage. As organizations move to shared storage, the cost of storage is a major concern.

Solution—Local Distributed File System: Local distributed file systems like Hadoop's HDFS leverage local disk space, which is about \$1.20/GB, to store data, thus reducing the cost of storage.

Problem—Cost of proprietary hardware: Proprietary hardware deployed to process data is expensive. Organizations are moving to open source and software license environments. Open source environments, or in million dollar technology in terabyte scale, are expensive.

Solution—Converged high-performance hardware: One vendor for a converged system for a \$1.2 million in licensing. The support costs were \$400,000 and support costs have been constantly growing in \$50 increments.

Problem—Complex proprietary system architecture: Proprietary system architecture for every 40 systems known.

Solution—Hardware as a platform: Numerous administrative hardware, stack.

Q 33 Write detailed note on Hadoop.

Ans. Hadoop : Doug Cutting, Mike Cafarella and team took the solution provided by Google and started an Open Source Project called HADOOP in 2005 and Doug named it after his son's toy elephant. Now Apache Hadoop is a registered trademark of the Apache Software Foundation. Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. In short, Hadoop framework is capable enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for a huge amounts of data.

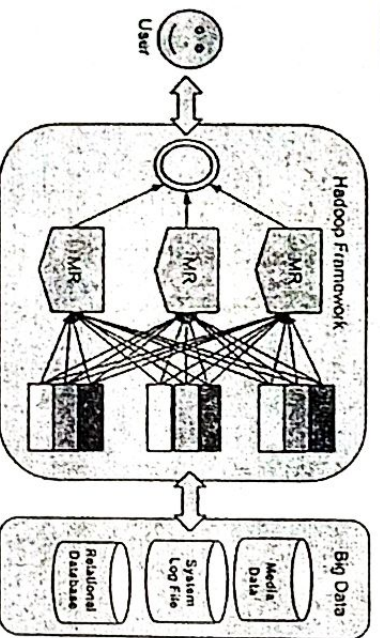


Fig.

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Today, we're surrounded by data. People upload videos, take pictures on their cell phones, text friends, update their Facebook status, leave comments around the web, click on ads, and so forth. Machines, too, are generating and keeping more and more data. The exponential growth of data first presented challenges to cutting-edge businesses such as Google, Yahoo, Amazon, and Microsoft. They needed to go through terabytes and petabytes of data to figure out which websites were popular, what books were in demand, and what kinds of ads appealed to people. Existing tools were becoming inadequate to process such large data sets.

Big Data Analytics

Google was the first to publicize MapReduce—a system they had used to scale their data processing needs. This system aroused a lot of interest because many other businesses were facing similar scaling challenges, and it wasn't feasible for everyone to reinvent their own proprietary tool.

Doug Cutting saw an opportunity and led the charge to develop an open source version of this MapReduce system called Hadoop. Soon after, Yahoo and others rallied around to support this effort. Today, Hadoop is a core part of the computing infrastructure for many web companies, such as Yahoo, Facebook, LinkedIn, and Twitter. Many more traditional businesses, such as media and telecom, are beginning to adopt this system too.

Hadoop, and large-scale distributed data processing in general, is rapidly becoming an important skill set for many programmers. An effective programmer, today, must have knowledge of relational databases, networking, and security, all of which were considered optional skills a couple decades ago. Similarly, basic understanding of distributed data processing will soon become an essential part of every programmer's toolbox. Leading universities, such as Stanford and CMU, have already started introducing Hadoop into their computer science curriculum.

Formally speaking, Hadoop is an open source framework for writing and running distributed applications that process large amounts of data. Distributed computing is a wide and varied field, but the key distinctions of Hadoop are that it is:

- Accessible :** Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon's Elastic Compute Cloud (EC2).
- Robust :** Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.
- Scalable :** Hadoop scales linearly to handle larger data by adding more nodes to the cluster.
- Simple :** Hadoop allows users to quickly write efficient parallel code.

Q 34 Write detailed note on GFS.

Ans. GFS : Google File System is a scalable distributed file system (DFS) created by Google Inc. and developed to

to store
ed, and
storage.
acy and
eys, and
limits an
s. Atomic
gregated
low cost
stored in
can then
to find
uracy of
because
s higher
big data
e models
building
ing new
ud have
not ideal.
ad went

acc
requ
scal
comm
built
optim
stora
huge
off-l
GFS
usage
enorm
File S
devel
of Go
divid
cluste
extrem
appen
on Go
of che
must b
and th

the first to publicize MapReduce—a tool to scale their data processing needs. It was a lot of interest because many other people were facing similar scaling challenges, and it was the first to publicize MapReduce—a tool to scale their data processing needs.

Google saw an opportunity and led the charge in developing a source version of this MapReduce system. Today, Yahoo and others rallied around the effort. Today, Hadoop is a core part of the infrastructure for many web companies, such as LinkedIn, and Twitter. Many more companies, such as media and telecom, are using this system too.

Large-scale distributed data processing is becoming an important skill set for many of the effective programmers, today, must have a solid understanding of distributed data processing. Large-scale distributed data processing is becoming an important skill set for many of the effective programmers, today, must have a solid understanding of distributed data processing.

speaking, Hadoop is an open source software and running distributed applications on a large number of commodity servers is a field, but the key distinction of Hadoop is that it is designed to run on commodity hardware.

Hadoop runs on large clusters of commodity machines or on cloud computing services like Amazon's Elastic Computer Cloud (EC2). Because it is intended to run on commodity hardware, Hadoop is architected with the assumption that hardware malfunctions. It can gracefully recover from such failures.

Hadoop scales linearly to handle larger data sets by adding more nodes to the cluster.

Hadoop allows users to quickly write efficient code.

distributed system on GFS

Google File System is a scalable distributed file system created by Google Inc. and developed to

accommodate Google's expanding data processing requirements. GFS provides fault tolerance, scalability, availability and performance to large networks of connected nodes. GFS is made up of several storage components built from low-cost commodity hardware components, optimized to accommodate Google's different data use and storage needs, such as its search engine, which generates huge amounts of data that must be stored.

The Google File System capitalized on the strength of off-the-shelf servers while minimizing hardware weaknesses. GFS is also known as GoogleFS.

GFS is enhanced for Google's core data storage and usage needs (primarily the search engine), which can generate enormous amounts of data that needs to be retained. Google File System grew out of an earlier Google effort, "Big Files", developed by Larry Page and Sergey Brin in the early days of Google, while it was still located in Stanford. Files are divided into fixed-size chunks of 64 megabytes, similar to clusters or sections in regular file systems, which are only extremely rarely overwritten, or struck. Files are usually appended to or read. It is also designed and optimized to run on Google's computing clusters, dense nodes which consist of cheap "commodity" computers, which means precautions must be taken against the high failure rate of individual nodes and the subsequent data loss.

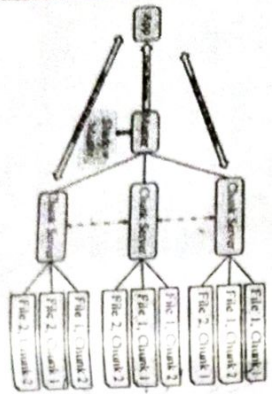


Fig.

A GFS cluster consists of multiple nodes. These nodes are divided into two types: One Master node and a large number of Chunk servers. Each file is divided into fixed-size chunks. Chunk servers store these chunks. Each chunk is assigned a unique 64-bit label by the master node at the time of creation, and logical mappings of files to constituent chunks are maintained. Each chunk is replicated several times throughout the network, with the minimum being three, but even more for files that have high end-in demand or need more redundancy.

BDA.18

The Master server does not usually store the actual chunks, but rather all the metadata associated with the chunks, such as the tables mapping the 64-bit labels to chunk locations and the files they make up, the locations of the copies of the chunks, what processes are reading or writing to a particular chunk, or taking a "snapshot" of the Master server, to replicate it (usually at the instigation of the Master server, when, due to node failures, the number of copies of a chunk has fallen beneath the get number). All this metadata is kept current by the Master server periodically receiving updates from each chunk server ("Heart-beat messages").

Permissions for modifications are handled by a system of time-limited, expiring "leases", where the Master server grants permission to a process for a finite period of time during which no other process will be granted permission by the Master server to modify the chunk. The modifying chunk server, which is always the primary chunk holder, then propagates the changes to the chunk servers with the backup copies. The changes are not saved until all chunk servers acknowledge, thus guaranteeing the completion and atomicity of the operation.

Programs access the chunks by first querying the Master server for the locations of the desired chunks; if the chunks are not being operated on (i.e. no outstanding leases exist), the Master replicates with the locations, and the program then contacts and receives the data from the chunk server directly (similar to Kazaa and its supernodes).

Unlike most other file systems, GFS is not implemented in the kernel of an operating system, but is instead provided as a user space library.

The GFS node cluster is a single master with multiple client systems. Chunk servers store data as Linux files on local disks. Stored data is divided into large chunks (64 MB), which are replicated in the network a minimum of three times. The large chunk size reduces network overhead.

GFS is designed to accommodate Google's large cluster requirements without burdening applications. Files are stored in hierarchical directories identified by path names. Metadata information is controlled by the master, which interacts with client systems. Chunk servers store data as Linux files on local disks. Stored data is divided into large chunks (64 MB), which are replicated in the network a minimum of three times. The large chunk size reduces network overhead.

GFS features include:

- (i) Fault tolerance
- (ii) Critical data replication

B.Tech. (VIII Sem.) C.S. Solved Papers

- (iii) Automatic and efficient data recovery
- (iv) High aggregate throughput
- (v) Reduced client and master interaction because of large chunk server size
- (vi) Namespace management and locking
- (vii) High availability

The largest GFS clusters have more than 1,000 nodes with 300 TB disk storage capacity. This can be accessed by hundreds of clients on a continuous basis.

Q.35 What is HDFS? Write its features, goals and advantages.

Ans. When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. File systems that manage the storage across a network of machines are called distributed file systems. Since they are network-based, all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk file systems. For example, one of the biggest challenges is making the file system tolerate node failure without suffering data loss. Hadoop comes with a distributed file system called HDFS, which stands for Hadoop Distributed File system.

Hadoop can work directly with any mountable distributed file system such as Local FS, HTTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

Big Data Analytics

HDFS provides a shell like any other list of commands are available to interact with the data.

Advantages of HDFS

- (i) Hadoop framework allows the automatic distributed systems. It automatically distributes the data machines and in turn, utilizes the CPU cores.
- (ii) Hadoop does not rely on hardware tolerance and high availability. (It library itself has been designed to handle failures at the application layer.)
- (iii) Servers can be added or removed dynamically and Hadoop continues to run without interruption.
- (iv) Another big advantage of Hadoop is being open source, it is compatible since it is Java based.

Hadoop File System was developed as a file system design. It is run on commodity hardware distributed systems, HDFS is designed using low-cost hardware. HDFS holds very large amount of data. To store such huge data across multiple machines. These files are stored in a distributed fashion to rescue the system from case of failure. HDFS also makes a parallel processing.

Features of HDFS

- (i) It is suitable for the distributed storage.
- (ii) Hadoop provides a command line interface.
- (iii) The built-in servers of NameNode users to easily check the status of the system.
- (iv) Streaming access to file system.
- (v) HDFS provides file permissions.

Goals of HDFS

- (i) Fault Detection and Recovery: HDFS includes a large number of components is frequent. It has mechanisms for quick detection and recovery.
- (ii) Huge Datasets: HDFS stores data in nodes per cluster to manage huge datasets.

chunk server. Modifications are handled by a system called *Master*, where the Master server issues for a process a finite period of time during which it is allowed to modify the chunk. The modification process will be granted permission by the Master server to modify the primary chunk holder, then the changes are not saved until all chunk servers agree, thus guaranteeing the completion and atomicity of the modification.

frames access the chunks by first querying the server for the locations of the desired chunks, if they not being operational (i.e. no outstanding leases). Master replies with the locations, and the program acquires and receives the data from the chunk server (similar to Kazaa and its supernodes).

ke most other file systems, GFS is not implemented as part of an operating system, but is instead provided as a user-space library.

GFS node cluster is a single master with multiple servers that are continuously accessed by different clients. Chunk servers store data as Linux files on local disks. Stored data is divided into large chunks (64 MB), replicated in the network a minimum of three times. Chunk size reduces network overhead.

is designed to resemble Google's large cluster of servers without building applications. Files are stored in a controlled, central location. Metadata is controlled by the user, which interacts with the status updated and chunk server through the messages. The features include:

- (iii) Automatic and efficient data recovery
- (iv) High aggregate throughput
- (v) Reduced client and master interaction because of large chunk server size
- (vi) Namespace management and locking
- (vii) High availability

The largest GFS clusters have more than 1,000 nodes with 300 TB disk storage capacity. This can be accessed by hundreds of clients on a continuous basis.

Q35) What is HDFS? Write its features, goals and advantages.

Ans. When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. File systems that manage the storage across a network of machines are called distributed file systems. Since they are network-based, all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk file systems. For example, one of the biggest challenges is making the file system tolerate node failure without suffering data loss. Hadoop comes with a distributed file system called HDFS, which stands for Hadoop Distributed File system.

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

The largest GFS clusters have more than 1,000 nodes with 300 TB disk storage capacity. This can be accessed by hundreds of clients on a continuous basis.

Q35 What is HDFS? Write its features, goals and advantages.

Ans. When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. File systems that manage the storage across a network of machines are called distributed file systems. Since they are network-based, all the complications of network programming kick in, thus making distributed file systems more complex than regular disk file systems. For example, one of the biggest challenges is making the file system tolerate node failure without suffering data loss. Hadoop comes with a distributed file system called HDFS, which stands for Hadoop Distributed File system.

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of `DataNodes`. The `NameNode` determines the mapping of blocks to the `DataNodes`. The `DataNodes` takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by `NameNode`.

(iii) **Hardware at Data** : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it

- (i) Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it

automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.

- (ii) Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.

- (iii) Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.

- (iv) Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available for parallel processing.

Features of HDFS

(i) It is suitable for the distributed storage and processing.

- (ii) Hadoop provides a command interface to interact with HDFS.

- (iii). The built-in servers of NameNode and DataNode help users to easily check the status of cluster.

- (iv) Streaming access to file system data

- (v) HDFS provides file permissions and authentication

Goals of HDPS

(i) **Fault Detection and Recovery :** Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

- (ii) **Huge Datasets** : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

- (iii) **Hardware at Data** : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

Q36 Explain building blocks of Hadoop in detail.

Ans. HDFS is responsible for storing data on the cluster in Hadoop. Files in HDFS are split into blocks before they are stored on cluster of size 64MB or 128MB. On a fully configured cluster, "running Hadoop" means running a set of daemons, or resident programs--on the different servers in your network. Programs which reside permanently in memory are called "Resident Programs". Daemon is a thread in Java, which runs in background and mostly created by JVM for performing background task like Garbage collection. Each daemon runs separately in its own JVM. These daemons have specific roles; some exist only on one server, some exist across multiple servers. The daemons include :

1. **NameNode**
2. **DataNode**
3. **Secondary NameNode**
4. **JobTracker**
5. **TaskTracker**

The above daemons are called as "Building Blocks of Hadoop".

1. **NameNode** : Let's begin with arguably the most vital of the Hadoop daemons—the NameNode. Hadoop employs a master/slave architecture for both distributed storage and distributed computation. The distributed storage system is called the Hadoop File System, or HDFS. The NameNode is the master of HDFS that directs the slave **DataNodes** daemons to perform the low-level I/O tasks. The NameNode is the bookkeeper of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system. The function of the NameNode is memory and I/O intensive; as such, the server hosting the NameNode typically doesn't store any user data or perform any computations for the MapReduce program to lower the workload on the machine. This means that the NameNode server doesn't double as a **DataNode** or a **TaskTracker**.

There is unfortunately a negative aspect to the importance of the NameNode—it's a single point of failure.

of your Hadoop cluster. For any of the other daemons, if their host nodes fail for software or hardware reasons, the Hadoop cluster will likely continue to function smoothly or you can quickly restart it. Not so for the NameNode.

2. **DataNode** : Each slave machine in your cluster will host a DataNode daemon to perform the grunt (thankless and menial) work of the distributed filesystem – reading and writing HDFS blocks to actual files on the local filesystem. When you want to read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in. Your client communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

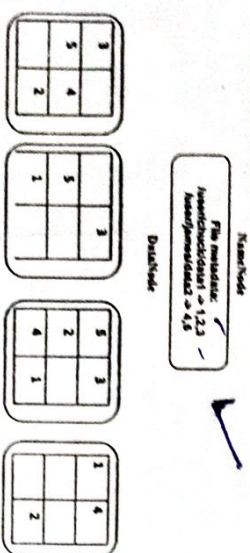


Fig. 1 : NameNode/DataNode interaction in HDFS

Fig. 1 illustrates the roles of the NameNode and DataNodes. In this figure, we show two data files, one at /user/chuck/data 1 and another at /user/james/data 2. The data1 file takes up three blocks, which we denote 1, 2, and 3, and the data2 file consists of blocks 4 and 5. The content of the files are distributed among the DataNodes.

In this illustration, each block has three replicas. For example, block 1 (used for data 1) is replicated over the three rightmost DataNodes. This ensures that if any one DataNode crashes or becomes inaccessible over the network, you'll still be able to read the files. DataNodes are constantly reporting to the NameNode. Upon initialization, each of the DataNodes informs the NameNode of the blocks it's currently storing. After this mapping is complete, the DataNodes continually poll the NameNode to provide information regarding local changes as well as receive instructions to create, move, or delete blocks from the local disk.

3. **Secondary NameNode** : The Secondary NameNode (SNN) is an assistant daemon for monitoring the state of the cluster HDFS. Like the NameNode, each cluster has one SNN, and it typically resides on its own machine as well. No other DataNode or TaskTracker daemons run on the same

server. The SNN differs from the NameNode in that this process doesn't receive or record any real-time changes to HDFS. Instead, it communicates with the NameNode to take snapshots of the HDFS metadata at intervals defined by the cluster configuration. The NameNode is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data. Nevertheless, a NameNode failure requires human intervention to reconfigure the cluster to use the SNN as the primary NameNode.

4. **JobTracker** : The JobTracker daemon is the liaison (communication/cooperation which facilitates a close working) between your application and Hadoop. Once you submit your code to your cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running. Should a task fail, the JobTracker will automatically relaunch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

5. **TaskTracker** : As with the storage daemons, the computing daemons also follow a master/slave architecture: the JobTracker is the master overseeing the overall execution of a MapReduce job and the TaskTracker manage the execution of individual tasks on each slave node.

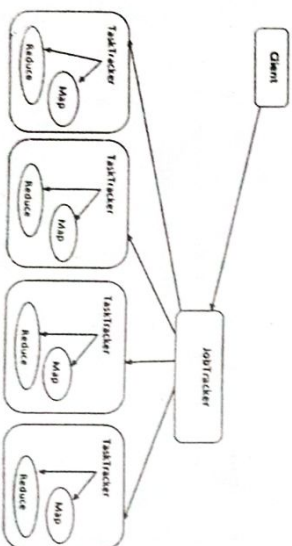


Fig. 2 : JobTracker and TaskTracker interaction

Fig. 2 illustrates this interaction. Each TaskTracker is responsible for executing the individual tasks that the JobTracker assigns. Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple JVMs to handle many map or reduce tasks in parallel. One responsibility of the TaskTracker is to constantly communicate with the JobTracker. If the JobTracker fails to receive a heartbeat from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.

Having covered each of the Hadoop daemons, we depict the topology of one typical Hadoop cluster in the fig. 3.

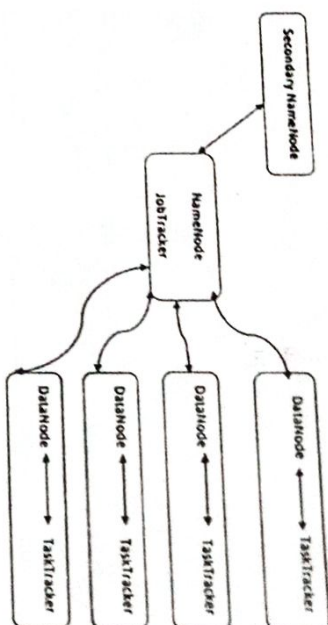


Fig. 3 : Topology of a typical Hadoop cluster

This topology features a master node running the NameNode and JobTracker daemons and a standalone node with the SNN in case the master node fails. For small clusters, the SNN can reside on one of the slave nodes. On the other hand, for large clusters, separate the NameNode and JobTracker on two machines. The slave machines each host a DataNode and TaskTracker, for running tasks on the same node where their data is stored.

Q.37 Explain process of configuring hadoop cluster and also describe various operational modes of Hadoop.

Ans. When setting up a Hadoop cluster, you'll need to designate one specific node as the master node. Server will typically host the NameNode and JobTracker daemons. It'll also serve as the base station-contacting and activating the DataNode and TaskTracker daemons on all of the slave nodes. As such, we need to define a means for the master node to remotely access every node in your cluster.

Hadoop uses passphrase less SSH for this purpose. SSH utilizes standard public key cryptography to create a pair of keys for user verification—one public, one private. The public key is stored locally on every node in the cluster, and the master node sends the private key when attempting to access a remote machine. With both pieces of information, the target machine can validate the login attempt.

Define a Common Account : We've been speaking in general terms of one node accessing another; more precisely this access is from a user account on one node to another user account on the target machine. For Hadoop, the accounts should have the same username on all of the nodes and for security purpose we recommend it being a `hadoop` user.

This account the cluster data your actual SSH is installed of the "which

[hadoop] /usr [hadoop] /usr [hadoop] /usr [hadoop] /usr

If you i /usr Install package mana Operational

1. Local (Stan default mode Hadoop source Hadoop choos configuration. version 0.20) a

<?xml version= <?xml-stylesheet= <!-- Put site-sp <configuration> </configuration>

With en completely on t communicate w use HDFS, nor primary use is f logic of a Ma complexity of i Properties:

(i) Default t (ii) HDFS is (iii) Local Fi (iv) Used for (v) Shared

mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>master:9001</value>
<description>The host and port that the MapReduce job
tracker runs
at.</description>
</property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>dfs.replication</name>
<value>3</value>
<description>The actual number of replications can be
specified when the file is created.</description>
</property>
</configuration>
```

The key differences are :

- (i) We explicitly stated the hostname for location of the NameNode and JobTracker daemons.
- (ii) We increased the HDFS replication factor to take advantage of distributed storage. Recall that data is replicated across HDFS to increase availability and reliability.

Properties:

- (i) This is a production phase.
- (ii) Data are used and distributed across many nodes.
- (iii) Different nodes will be used as Master node/Data node etc.

Q.38 Explain how to configure xml files?

Ans.

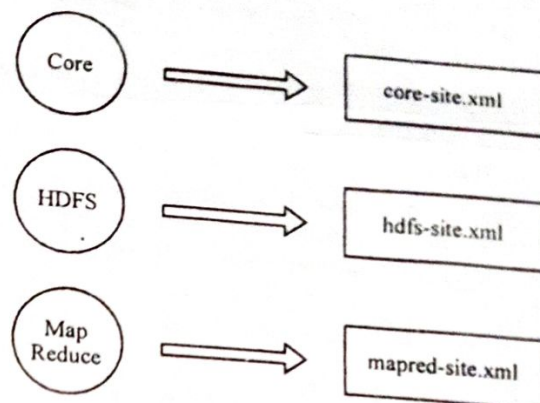


Fig.

Hadoop Cluster Configuration Files :

Configuration Filenames	Description of Log Files
Hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.
hdfs-site.xml	Configuration settings for HDFS daemons, the namenode, the secondary namenode and the data nodes.
mapred-site.xml	Configuration settings for MapReduce daemons; the job-tracker and the task-trackers.
masters	A list of machines (one per line) that each run a secondary namenode.
slaves	A list of machines (one per line) that each run a datanode and a task-tracker.

All these files are available under 'conf' directory of Hadoop installation directory.

Here is a listing of these files in the File System:

```

ubuntu@ip-10-251-81-223:~/hadoop-1.2.0$ cd conf/
ubuntu@ip-10-251-81-223:~/hadoop-1.2.0/conf$ ls
capacity-scheduler.xml  hadoop-policy.xml  slaves
configuration.xml      hdfs-site.xml     xsl-client.xml
core-site.xml          log4j.properties  xsl-server.xml
fair-scheduler.xml     mapred-queue-decl.xml  taskcontroller.cfg
hadoop-env.sh          mapred-site.xml    task-log4j.properties
hadoop-metrics2.properties  master
ubuntu@ip-10-251-81-223:~/hadoop-1.2.0/conf$
```

Fig.

Let's look at the files and their usage one by one!

hadoop-env.sh

This file specifies environment variables that affect the JDK used by Hadoop.

Daemon (bin/hadoop)

As Hadoop framework is written in Java and uses Java Runtime environment, one of the important environment variables for Hadoop daemon is \$JAVA_HOME in hadoop-env.sh.

This variable directs Hadoop daemon to the Java path in the system.

The Java implementation to use. Required

```
export JAVA_HOME = /usr/lib/jvm/java-1.6.0-openjdk-amd64
```

This file is also used for setting another Hadoop daemon execution environment such as heap size (HADOOP_HEAP), hadoop home (HADOOP_HOME), log file location (HADOOP_LOG_DIR), etc.

Note: For the simplicity of understanding the cluster setup, we have configured only necessary parameters to start a cluster.

The following three files are the important configuration files for the runtime environment settings of a Hadoop cluster.

core-site.sh

This file informs Hadoop daemon where NameNode runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.

```
<?xml version="1.0"?>
```

```
<?xml:stylesheet type="text/xml" href="configuration.xml"?>
```

```
<!-- Put site-specific property overrides in this file -->
```

```
<configuration>
```

```
<property>
```

```
<name>fs.default.name</name>
```

```
<value>hdfs://ec2-54-214-206-65.us-west-2.compute.amazonaws.com:8020</value>
```

```
</property>
```

```
</configuration>
```

Where hostname and port are the machine and port on which NameNode daemon runs and listens. It also informs the Name Node as to which IP and port it should bind. The

commonly used port is 8020 and you can also specify IP address rather than hostname.

hdfs-site.sh

This file contains the configuration settings for HDFS daemons; the Name Node, the Secondary Name Node, and the data nodes.

You can also configure hdfs-site.xml to specify default block replication and permission checking on HDFS. The actual number of replications can also be specified when the file is created. The default is used if replication is not specified in create time.

```
<?xml version="1.0"?>
```

```
<?xml:stylesheet type="text/xml" href="configuration.xml"?>
```

```
<!-- Put site-specific property overrides in this file -->
```

```
<configuration>
```

```
<property>
```

```
<name>dfs.replication</name>
```

```
<value>3</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.permissions</name>
```

```
<value>false</value>
```

```
</property>
```

```
</configuration>
```

The value "true" for property 'dfs.permissions' enables permission checking in HDFS and the value "false" turns off the permission checking. Switching from one parameter value to the other does not change the mode, owner or group of files or directories.

mapred-site.sh

This file contains the configuration settings for MapReduce daemons; the job tracker and the task-trackers.

The mapred.job.tracker parameter is a hostname (or IP address) and port pair on which the Job Tracker listens for RTR communication. This parameter specifies the location of the job tracker to Task Trackers and MapReduce clients.

```
<?xml version="1.0"?>
```

```
<?xml:stylesheet type="text/xml" href="configuration.xml"?>
```

```
<!-- Put site-specific property overrides in this file -->
```

```
<configuration>
```

```
<property>
```

```
<name>mapred.job.tracker</name>
```

```
<value>ec2-54-214-206-65.us-west-2.compute.amazonaws.com:8021</value>
```

```
</property>
```

```
</configuration>
```

You can replicate all of the four files explained above to all the Data Nodes and Secondary NameNode. These files can then be configured for any node specific configuration e.g. in case of a different JAVA_HOME on one of the DataNodes.

The following two files 'masters' and 'slaves' determine the master and slave Nodes in Hadoop cluster.

Masters : This file informs about the Secondary NameNode location to Hadoop daemon. The 'masters' file at Master server contains a hostname Secondary Name Node servers.

Slaves : Contains a list of hosts, one per line, that are to host DataNode and TaskTracker servers.

Masters : Contains a list of hosts, one per line, that are to host Secondary NameNode servers.

The 'masters' file on Slave Nodes is blank.

Slaves : The 'slaves' file at Master node contains a list of hosts, one per line, that are to host Data Node and TaskTracker servers.

```
ec2-54-218-170-127.us-west-2.compute.amazonaws.com
ec2-54-202-24-115.us-west-2.compute.amazonaws.com
```

The 'slaves' file on Slave server contains the IP address of the slave node. Notice that the 'slaves' file at Slave node contains only its own IP address and not of any other Data Nodes in the cluster.

ed cache in MapReduce

important feature provided by
when you want to share some
cluster, distributed cache is
cutable jar files or simple

r different from chain

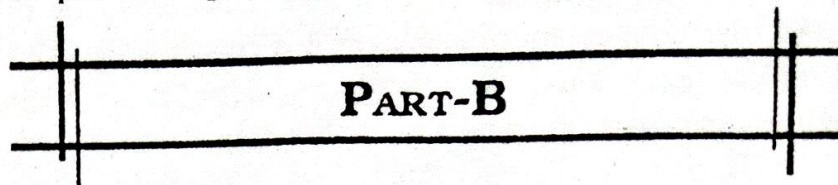
Chain Mapper
This class is used to run multiple mappers in a single map task.
The output of the first mapper becomes the input to the second mapper, second to third and so on.
It is defined in : org.apache.Hadoop. mapreduce.lib.chain. ChainMapperpackage

ds of a Reducer.

hods of a reducer. They are :
configure different parameters
ed cache and input data.
er that is called once per key
uce task.
emporary files and called only
task.

The combiner receives data from the map tasks, works on it, and then passes its output to the reducer phase.

Partitioner : The partitioner decides how many reduced tasks would be used to summarize the data. It also confirms how outputs from combiners are sent to the reducer, and controls the partitioning of keys of the intermediate map outputs.



Q.14 Explain MapReduce program with neat figure.

Ans. Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

The term MapReduce actually refers to the following two different tasks that Hadoop programs perform:

- **The Map Task :** Refer to Q.1.
- **The Reduce Task :** Refer to Q.2.

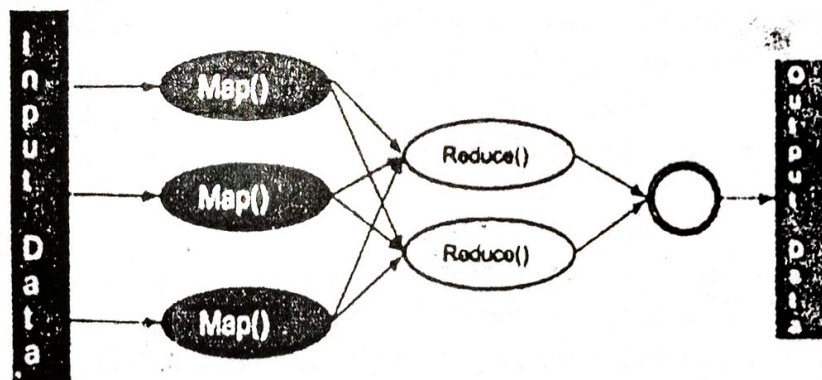


Fig.

Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for resource management, tracking

BDA.28

resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically. The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster. The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

Most of the computing takes place on nodes with data on local disks that reduces the network traffic. After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Q.15 Explain the process of analyzing the data with Hadoop.

Our map
the air temper
interested in.
preparation ph
reducer functio
temperature fo
place to drop
that are missin

To visu
following sam
have been drop

006701199099999
004301199099999
004301199099999
004301265099999
004301265099999

These l
key-value pair

(0, 006701199099999
(106, 004301199099999
(212, 004301199099999
(318, 004301265099999
(424, 004301265099999

The ke
ignore in our
the year and t
emits them as
interpreted as

(1
(1
(1

Q.16 Explain the various difference between old and new java MapReduce API.

Ans.

Difference	New API	Old API
Mapper and Reducer	New API using Mapper and Reducer as Class. So can add a method (with a default implementation) to an abstract class without breaking old implementations of the class	In Old API used Mapper and Reducer as Interface (still exist in New API as well)
Package	New API is in the org.apache.hadoop.mapreduce package	Old API can still be found in org.apache.hadoop.mapred.
User Code to communicate with MapReduce System	Use "context" object to communicate with MapReduce system	JobConf, the OutputCollector, and the Reporter object use for communicate with MapReduce System
Control Mapper and Reducer execution	New API allows both mappers and reducers to control the execution flow by overriding the run() method.	Controlling mappers by writing a MapRunnable, but no equivalent exists for reducers.
Job Control	Job control is done through the Job class in New API	Job control was done through JobClient (not exists in the new API)
Job Configuration	Job configuration done through configuration class via some of the helper methods on Job.	jobconf object was use for Job configuration, which is extension of configuration class. java.lang.Object extended by org.apache.hadoop.conf. Configuration extended by org.apache.hadoop.mapred.JobConf
OutPut File Name	In the new API map outputs are named part-m-nnnnn, and reduce outputs are named part-r-nnnnn (where nnnnn is an integer designating the part number, starting from zero).	In the old API both map and reduce outputs are named part-nnnnn
reduce () method passes values	In the new API, the reduce() method passes values as a java.lang.Iterable	In the old API, the reduce() method passes values as a java.lang.Iterator

Q.17 Explain mapper. How it works?

Ans. Mapper : To serve as the mapper, a class implements from the Mapper interface and inherits the MapReduceBase class. The MapReduceBase class, not surprisingly, serves as the base class for both mappers and reducers. It includes two methods that effectively act as the constructor and destructor for the class:

void configure(JobConf job) — In this function you can extract the parameters set either by the configuration XML files or in the main class of your application. Call this function before any data processing begins.

void close () — As the last action before the map task terminates, this function should wrap up any loose ends—database connections, open files, and so on.

The Mapper interface is responsible for the data processing step. It utilizes Java generics of the form $\text{Mapper}\langle K_1, V_1, K_2, V_2 \rangle$ where the key classes and value classes implement the WritableComparable and Writable interfaces, respectively. Its single method is to process an individual (key/value) pair:

```
void map(K1 key,
        V1 value,
        OutputCollector<K2,V2> output,
        Reporter reporter
        ) throws IOException
```

The function generates a (possibly empty) list of (K_2, V_2) pairs for a given (K_1, V_1) input pair. The OutputCollector receives the output of the mapping process, and the Reporter provides the option to record extra information about the mapper as the task progresses. Hadoop provides a few useful mapper implementations. You can see some of them in the table.

Class	Description
IdentityMapper<K,V>	Implements Mapper<K,V,K,V> and maps inputs directly to outputs
InverseMapper<K, V>	Implements Mapper<K,V,V,K> and reverses the key/value pair
RegexMapper<K>	Implements Mapper<K, Text, Text, Long Writable> and generates a (match, 1) pair for every regular expression match

TokenCountMapper<K>

Implements Mapper<K, Text, Text, Long Writable> and generates a (token, 1) pair when the input value is tokenized

Q.18 Define combiner.

Ans. Combiner : A combiner, also known as a semi-reducer, is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class. The main function of a combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.

The combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

The following MapReduce task diagram shows the combiner phase.

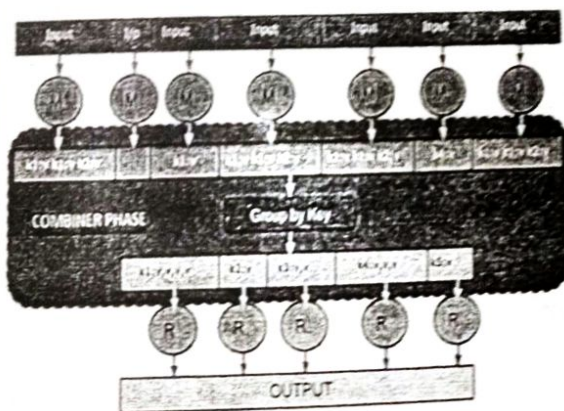


Fig.

Here is a brief summary on how MapReduce combiner works :

- A combiner does not have a predefined interface and it must implement the Reducer interface's reduce() method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.

(iii) A combiner can produce summary information from a large dataset because it replaces the original Map output.

Although, combiner is optional yet it helps segregating data into multiple groups for Reduce phase, which makes it easier to process.

Q.19 Explain how MapReduce work diagrammatically with example.

Ans. In MapReduce, during the map phase, it counts the words in each document, while in the reduce phase it aggregates the data as per the document spanning the entire collection. During the map phase, the input data is divided into splits for analysis by map tasks running in parallel across Hadoop framework.

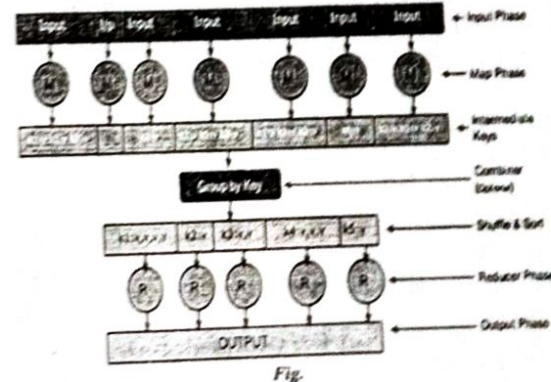


Fig.

Example

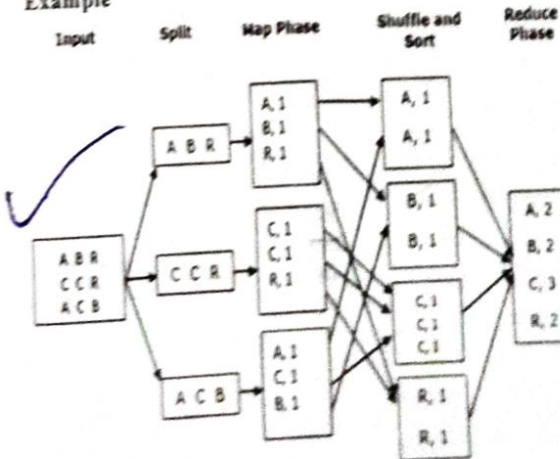


Fig.

The script loops through the compressed year files, first printing the year, and then processing each file using *awk*. The *awk* script extracts two fields from the data: The air temperature and the quality code. The air temperature value is turned into an integer by adding 0. Next, a test is applied to see if the temperature is valid (the value 9999 signifies a missing value in the NCDC dataset) and if the quality code indicates that the reading is not suspect or erroneous. If the reading is OK, the value is compared with the maximum value seen so far, which is updated if a new maximum is found. The END block is executed after all the lines in the file have been processed, and it prints the maximum value.

Here is the beginning of a run:

```
% ./max_temperature.sh
```

```
1901 317
```

```
1902 244
```

```
1903 289
```

```
1904 256
```

```
1905 283
```

```
---
```

The temperature values in the source file are scaled by a factor of 10, so this works out as a maximum temperature of 31.7°C for 1901 (there were very few readings at the beginning of the century, so this is plausible). The complete run for the century took 42 minutes in one run on a single EC2 High-CPU Extra Large Instance.

Q.22 Write notes on followings :

(a) Reducer

(b) Partitioner

Ans.(a) Reducer : As with any mapper implementation, a reducer must first extend the MapReduce base class to allow for configuration and cleanup. In addition, it must also implement the Reducer interface which has the following single method:

```
void reduce{K2 key,
```

```
Iterator<V2> values,
```

```
OutputCollector<K3,V3> output,
```

```
Reporter reporter
```

```
} throws IOException
```

When the reducer task receives the output from the various mappers, it sorts the incoming data on the key of the (key/value) pair and groups together all values of the same key. The `reduce()` function is then called, and it generates a (possibly empty) list of (K3, V3) pairs by iterating over the values associated with a given key. The `OutputCollector` receives the output of the reduce process and writes it to an output file. The `Reporter` provides the option to record extra information about the reducer as the task progresses.

The below table lists a couple of basic reducer implementations provided by Hadoop.

Class	Description
IdentityReducer <K, V>	Implements Reducer <K, V, K, V> and maps inputs directly to outputs
LongSumReducer<K>	Implements Reducer <K, LongWritable, K, LongWritable> and determines the sum of all values corresponding to the given key

Although we have referred to Hadoop programs as MapReduce applications, there is a vital step between the two stages: Directing the result of the mappers to the different reducers. This is the responsibility of the partitioner.

Ans.(b) Partitioner : A common misconception for first-time MapReduce programmers is to use only a single reducer. After all, a single reducer sorts all of your data before processing—and who doesn't like sorted data? Our discussions regarding MapReduce expose the folly of such thinking. We would have ignored the benefits of parallel computation. With one reducer, our compute cloud has been demoted to a compute raindrop.

With multiple reducers, we need some way to determine the appropriate one to send a (key/value) pair outputted by a mapper. The default behavior is to hash the key to determine

Big Data Analytics

The mapping and reducing functions are identified by the `setMapperClass()` and `setReducerClass()` methods. The data types emitted by the reducer are identified by `setOutputKeyClass()` and `setOutputValueClass()`. By default, it is assumed that these are the output types of the mapper as well. If this is not the case, the methods `setMapOutputKeyClass()` and `setMapOutputValueClass()` methods of the `JobConf` class will override these. The input types fed to the mapper are controlled by the `InputFormat` used.

The default input format, "TextInputFormat," will load data in as (LongWritable, Text) pairs. The long value is the byte offset of the line in the file. The text object holds the string contents of the line of the file.

The call to `JobClient.runJob(conf)` will submit the job to MapReduce. This call will block until the job completes. If the job fails, it will throw an `IOException`. `JobClient` also provides a non-blocking version called `submitJob()`.

Q.24 Explain RecordReader in detail.

Ans. RecordReader : The `RecordReader` class actually loads the data from its source and converts it into (key, value) pairs suitable for reading by the Mapper. The `RecordReader` instance is defined by the `InputFormat`. The default `InputFormat`, `TextInputFormat`, provides a `LineRecordReader`, which treats each line of the input file as a new value. The key associated with each line is its byte offset in the file. The `RecordReader` is invoked repeatedly on the input until the entire `InputSplit` has been consumed. Each invocation of the `RecordReader` leads to another call to the `map()` method of the Mapper.

In the context of file-based input, the "start" is the byte position in the file where the `RecordReader` should start generating key/value pairs. The "end" is where it should stop reading records. These are not hard boundaries as far as the API is concerned—there is nothing stopping a developer from reading the entire file for each map task. While reading the entire file is not advised, reading outside of the boundaries is often necessary to ensure that a complete record is generated.

Example (LineRecordReader) : Let's get an example that will hopefully make the code slightly more readable. Suppose my data set is composed on a single 300Mb file, spanned over 3 different blocks (blocks of 128Mb). Suppose that I have been able to get 1 `InputSplit` for a block. Let's imagine now 3 different scenarios.

File is composed on 6 lines of 50 Mb each as shown below :

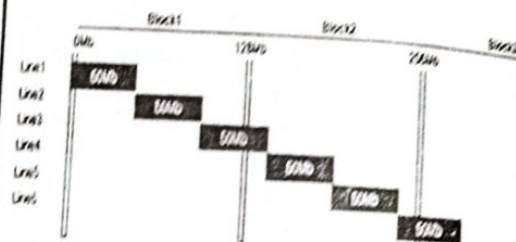


Fig.

The first Reader will start reading bytes from Block B1, position 0. The first two EOL will be met at respective 50Mb and 100Mb. 2 lines (L1 & L2) will be read and sent as key / value pairs to Mapper 1 instance. Then, starting from byte 100Mb, we will reach end of our Split (128Mb) before having found the third EOL. This incomplete line will be completed by reading the bytes in Block B2 until position 150Mb. First part of Line L3 will be read locally from Block B1, second part will be read remotely from Block B2 (by means of `FSDataInputStream`), and a complete record will finally be sent as key / value to Mapper 1.

The second Reader starts on Block B2, at position 128Mb. Because 128Mb is not the start of a file, there is a strong chance our pointer is located somewhere in an existing record that has been already processed by previous Readers. We need to skip this record by jumping out to the next available EOL, found at position 150Mb. Actual start of `RecordReader 2` will be at 150Mb instead of 128Mb.

We can wonder what happens in case a block starts exactly on a EOL. By jumping out until the next available record (through `readLine` method), we might miss 1 record. Before jumping to next EOL, we actually need to decrement initial "start" value to "start - 1". Being located at least offset before EOL; we ensure no record is skipped!

Remaining process is following same logic, and everything is summarized in below table.

	Start	Actual Start	End	Line(s)
Mapper1	B1:0	B1:0	B2:150	L1,L2,L3
Mapper2	B2:128	B2:150	B3:300	L4,L5,L6
Mapper3	B3:256	B3:300	B3:300	N/A

25 Explain MapReduce in detail.

MapReduce : The MapReduce paradigm provides a means to break a large task into smaller tasks, run the tasks in parallel, and consolidate the outputs of the individual tasks into the final output. As its name implies, MapReduce consists of two basic parts—a map step and a reduce step—tailored as follows:

Map:

- Applies an operation to a piece of data
- Provides some intermediate output

Reduce:

- Consolidates the intermediate outputs from the map steps
- Provides the final output

Each step uses key/value pairs, denoted as <key, value>, as input and output. It is useful to think of the key/value pairs as a simple ordered pair. However, the pairs can take fairly complex forms. For example, the key could be a filename, and the value could be the entire contents of the file.

The simplest illustration of MapReduce is a word count example in which the task is to simply count the number of times each word appears in a collection of documents. In practice, the objective of such an exercise is to establish a list of words and their frequency for purposes of search or establishing the relative importance of certain words. Figure illustrates the MapReduce processing for a single input—in this case, a line of text.

<1234, "For each word in each string">

↓ Map

<For, 1> <each, 1> <word, 1> <in, 1> <each, 1> <string, 1>

↓ Reduce

<For, 1>

<each, 2>

<word, 1>

<in, 1>

<string, 1>

Fig.

In this example, the map step parses the provided text string into individual words and emits a set of key/value pairs of the form <word, 1>. For each unique key—in this example, word—the reduce step sums the 1 values and outputs the <word, count > key/value pairs. Because the word each appeared twice in the given line of text, the reduce step provides a corresponding key/value pair of <each, 2>.

It should be noted that, in this example, the original key, 1234, is ignored in the processing. In a typical word count application, the map step may be applied to millions of lines of text, and the reduce step will summarize the key/value pairs generated by all the map steps.

Expanding on the word count example, the final output of a MapReduce process applied to a set of documents might have the key as an ordered pair and the value as an ordered tuple of length 2n. A possible representation of such a key/value pair follows:

<(filename, datetime), (word1, 5, word2, 7,..., wordn, 6)>

In this construction, the key is the ordered pair filename and datetime. The value consists of the n pairs of the words and their individual counts in the corresponding file.

Of course, a word count problem could be addressed in many ways other than MapReduce. However, MapReduce has the advantage of being able to distribute the workload over a cluster of computers and run the tasks in parallel. In a word count, the documents, or even pieces of the documents, could be processed simultaneously during the map step. A key characteristic of MapReduce is that the processing of one portion of the input can be carried out independently of

the processing of the other inputs. Thus, the workload can be easily distributed over a cluster of machines.

U.S. Navy rear admiral Grace Hopper (1906-1992), who was a pioneer in the field of computers, provided one of the best explanations of the need for using a group of computers. She commented that during pre-industrial times, oxen were used for heavy pulling, but when one ox couldn't budge a log, people didn't try to raise a larger ox; they added more oxen. Her point was that as computational problems grow, instead of building a bigger, more powerful, and more expensive computer, a better alternative is to build a system of computers to share the workload. Thus, in the MapReduce context, a large processing task would be distributed across many computers.

Although the concept of MapReduce has existed for

decades, Google led the resurgence in its interest and adoption starting in 2004 with the published work by Dean and Ghemawat. MapReduce has been used in functional programming languages such as Lisp, which obtained its name from being readily able to process lists (List processing).

In 2007, a well-publicized MapReduce use case was the conversion of 11 million New York Times news-paper articles from 1851 to 1980 into PDF files. The intent was to make the PDF files openly available to users on the Internet. After some development and testing of the MapReduce code on a local machine, the 11 million PDF files were generated on a 100-node cluster in about 24 hours.

What allowed the development of the MapReduce code and its execution to proceed easily was that the MapReduce paradigm had already been implemented in Apache Hadoop.

□□□

HADOOP I/O 3

IMPORTANT QUESTIONS

PART-A

Q.1 What is writable?

Ans. Hadoop uses its own serialization format, Writables, which is certainly compact and fast, but not so easy to extend or use from languages other than Java.

Q.2 What is serialization?

Ans. Serialization is the process of converting object data into byte stream data for transmission over a network across different nodes in a cluster or for persistent data storage.

Q.3 Define deserialization?

Ans. Deserialization is the reverse process of serialization and converts byte stream data into object data for reading data from HDFS. Hadoop provides Writables for serialization and deserialization purpose.

Q.4 Write the specification of writable interface.

Ans.
 package org.apache.hadoop.io;
 import java.io.DataInput;
 import java.io.DataOutput;
 import java.io.IOException;
 public interface Writable

```
{
    void write(DataOutput out) throws IOException;
    void readFields(DataInput in) throws IOException;
}
```

Q.5 Write the specification of Writable Comparable.

Ans. WritableComparable interface is sub-interface of Hadoop's Writable and Java's Comparable interfaces and its specification is shown below:

Comparable

```
{
    }
    The standard java.lang.Comparable interface contains single method compareTo() method for comparing the operators passed to it.
```

```
public interface Comparable
{
    public int compareTo(Object obj);
}
```

The compareTo() method returns -1, 0, or 1 depending on whether the compared object is less than, equal to, or greater than the current object.

Q.6 How is Hadoop related to Big Data?

Ans. Hadoop is an open-source framework for storing, processing, and analyzing complex unstructured data sets for deriving insights and intelligence.

Big Data Analytics

Q.7 Why do we need Hadoop for Big Data Analytics?

Ans. In most cases, Hadoop helps in exploring and analyzing large and unstructured data sets. Hadoop offers storage, processing and data collection capabilities that help in analytics.

Q.8 Explain the different features of Hadoop.

Ans. Open-source : Hadoop is an open-sourced platform. It allows the code to be rewritten or modified according to user and analytics requirements.

Scalability : Hadoop supports the addition of hardware resources to the new nodes.

Data Recovery : Hadoop follows replication which allows the recovery of data in the case of any failure.

Data Locality : This means that Hadoop moves the computation to the data and not the other way round. This way, the whole process speeds up.

Q.9 How WritableComparable can be implemented in Hadoop?

Ans. The implementation of WritableComparable is similar to Writable but with an additional 'compareTo' method inside it.

```
public interface WritableComparable extends Writable,
    comparable
{
    void readFields(DataInput in);
    void write(DataOutput out);
    int compareTo(WritableComparable o)
}
```

PART-B

Q.10 Write short note on Hadoop I/O.

Ans. Hadoop comes with a set of primitives for data I/O. Some of these are techniques that are more general than Hadoop, such as data integrity and compression, but deserve special consideration when dealing with multiterabyte

datasets. Others are Hadoop tools or APIs that are building blocks for developing distributed systems, such as serialization frameworks and on-disk data structures.

Data Integrity

Users of Hadoop rightly expect that no data will be corrupted during storage or processing. However, any I/O operation on the disk or network carries with it a chance of introducing errors into the data that is read or written, when the volumes of data flowing through the system are as large as the ones Hadoop is capable of handling. The chance of data corruption occurring is high.

The usual way of detecting corrupted data is by computing a checksum for the data when it first enters the system, and again whenever it is transmitted across a network. That is, the data is deemed to be corrupt if the newly generated checksum doesn't exactly match the original. This technique doesn't offer any way to fix the data—merely error detection. (And this is a reason for not using low-end hardware for Hadoop. In particular, be sure to use ECC memory.) Note that it's not the checksum that is corrupt, not the data, but the very unlikely, since the checksum is much smaller than the data.

A commonly used error-detecting code is a cyclic redundancy check (CRC), which computes a 32-bit checksum for input of any size.

Data Integrity in HDFS

HDFS transparently checksums all data written to the system and by default verifies checksums when reading data. A checksum is created for every 10 bytes per default block of data. The default is 512 bytes, and since the checksum is 4 bytes long, the storage overhead is 1%.

Datanodes are responsible for verifying the checksums received before storing the data and its checksum to the data that they receive from clients and for verifying the checksums during replication. A client writing data to a pipeline of datanodes, and the last datanode in the pipeline verifies the checksum. If it detects an error, the client throws a ChecksumException, a subclass of IOException, which should handle in an application-specific manner, such as the operation, for example.

When clients read data from datanodes, they verify the checksums as well, comparing them with the checksums stored in the datanode. Each datanode keeps a persistent checksum verification, so it knows the last time the data was verified.

blocks was verified. When a client successfully verifies a block, it tells the datanode, which updates its log. Keeping statistics such as these is valuable in detecting bad disks.

Aside from block verification on client reads, each datanode runs a DataBlockScanner in a background thread that periodically verifies all the blocks stored on the datanode. This is to guard against corruption due to "bit rot" in the physical storage media.

Since HDFS stores replicas of blocks, it can "heal" corrupted blocks by copying one of the good replicas to produce a new, uncorrupt replica. The way this works is that if a client detects an error when reading a block, it reports the bad block and the datanode it was trying to read from to the namenode before throwing a ChecksumException. The namenode marks the block replica as corrupt, so it doesn't direct clients to it, or try to copy this replica to another datanode. It then schedules a copy of the block to be replicated on another datanode, so its replication factor is back at the expected level. Once this has happened, the corrupt replica is deleted.

Q.11 Explain the following:

(a) localfilesystem

(b) checksumfilesystem

Ans.(a) LocalFileSystem : The Hadoop LocalFileSystem performs client-side checksumming. This means that when you write a file called filename, the filesystem client transparently creates a hidden file, filename.crc, in the same directory containing the checksums for each chunk of the file. Like HDFS, the chunk size is controlled by the 10 bytes per checksum property, which defaults to 512 bytes. The chunk size is stored as metadata in the .crc file, so the file can be read back correctly even if the setting for the chunk size has changed. Checksums are verified when the file is read, and if an error is detected, LocalFileSystem throws a ChecksumException.

Checksums are fairly cheap to compute (in Java, they are implemented in native code), typically adding a few percent overhead to the time to read or write a file. For most applications, this is an acceptable price to pay for data integrity. It is, however, possible to disable checksums: typically when the underlying filesystem supports checksums natively. This is accomplished by using RawLocalFileSystem in place of LocalFileSystem. To do this globally in an application, it suffices to remap the implementation for file URIs by setting

the property fs.file.impl to the value org.apache.hadoop.fs.RawLocalFileSystem. Alternatively, you can directly create a Raw LocalFileSystem instance, which may be useful if you want to disable checksum verification for only some reads; for example:

```
Configuration conf = ...
FileSystem fs = new RawLocalFileSystem();
fs.initialize(null, conf);
Configuration conf = ...
FileSystem fs = new RawLocalFileSystem();
fs.initialize(null, conf);
```

Ans.(b) ChecksumFileSystem

LocalFileSystem uses ChecksumFileSystem to do its work, and this class makes it easy to add checksumming to other (nonchecksummed) filesystems, as ChecksumFileSystem is just a wrapper around FileSystem. The general idiom is as follows:

```
FileSystem rawFs = ...
FileSystem checksummedFs = new
ChecksumFileSystem(rawFs);
```

The underlying filesystem is called the raw filesystem, and may be retrieved using the getRawFileSystem() method on ChecksumFileSystem. ChecksumFileSystem has a few more useful methods for working with checksums, such as getChecksumFile() for getting the path of a checksum file for any file. Check the documentation for the others. If an error is detected by ChecksumFileSystem when reading a file, it will call its reportChecksumFailure() method. The default implementation does nothing, but LocalFileSystem moves the offending file and its checksum to a side directory on the same device called bad_files. Administrators should periodically check for these bad files and take action on them.

Q.12 Explain writable collections.

Ans. Writable collections

There are six Writable collection types in the org.apache.hadoop.io package: Array Writable, Array PrimitiveWritable, TwoArrayWritable, MapWritable, Sorted MapWritable, and EnumSetWritable.

ArrayWritable and TwoArrayWritable are Writable implementations for arrays and two-dimensional arrays (array of arrays) of Writable instances. All the elements of an ArrayWritable or a TwoArrayWritable must be instances

of the same class, which is specified at construction, as follows:

```
ArrayWritable writable = new ArrayWritable(Text
class);
```

In contexts where the Writable is defined by type, such as in SequenceFile keys or values, or as input to MapReduce in general, you need to subclass ArrayWritable (or TwoDArrayWritable, as appropriate) to set the type statically. For example:

```
public class TextArrayWritable extends ArrayWritable {
    public TextArrayWritable() {
        super(Text.class);
    }
}
```

ArrayWritable and TwoDArrayWritable both have get() and set() methods, as well as a toArray() method, which creates a shallow copy of the array (or 2D array).

ArrayPrimitiveWritable is a wrapper for arrays of Java primitives. The component type is detected when you call set(), so there is no need to subclass to set the type.

MapWritable and SortedMapWritable are implementations of Java.util.Map<Writable, Writable> and Java.util.SortedMap<WritableComparable, Writable>, respectively. The type of each key and value field is a part of the serialization format for that field. The type is stored as a single byte that acts as an index into an array of types. The array is populated with the standard types in the org.apache.hadoop.io package, but custom Writable types are accommodated, too, by writing a header that encodes the type array for nonstandard types. As they are implemented, MapWritable and SortedMapWritable use positive byte values for custom types, so a maximum of 127 distinct nonstandard Writable classes can be used in any particular MapWritable or SortedMapWritable instance. Here's a demonstration of using a MapWritable with different types for keys and values:

```
MapWritable src = new MapWritable();
src.put(new IntWritable(1), new Text("cat"));
src.put(new VIntWritable(2), new LongWritable(163));
MapWritable dest = new MapWritable();
WritableUtils.cloneInto(dest, src);
assertThat((Text) dest.get(new IntWritable(1)), is(new
Text("cat")));
assertThat ((LongWritable) dest.get(new
```

```
VIntWritable(2)), is(new
```

```
LongWritable(163)));
```

Conspicuous by their absence are Writable collection implementations for sets and lists. A general set can be emulated by using a MapWritable (or a SortedMapWritable for a sorted set), with NullWritable values. There is also EnumSetWritable for sets of enum types. For lists of a single type of Writable, ArrayWritable is adequate, but to store different types of Writable in a single list, you can use GenericWritable to wrap the elements in an ArrayWritable. Alternatively, you could write a general ListWritable using the ideas from MapWritable.

Q.13 Explain custom comparator.

Ans. Custom Comparators : Writing raw comparators takes some care, since you have to deal with details at the byte level. It is worth looking at some of the implementations of Writable in the org.apache.hadoop.io package for further ideas, if you need to write your own. The utility methods on WritableUtils are very handy, too.

Custom comparators should also be written to be RawComparators, if possible. These are comparators that implement a different sort order to the natural sort order defined by the default comparator. Example shows a comparator for TextPair, called First Comparator, that considers only the first string of the pair. Note that we override the compare() method that takes objects so both compare() methods have the same semantics.

Example : A custom RawComparator for comparing the first field of TextPair byte representations

Solution :

```
public static class FirstComparator extends
WritableComparator {
    private static final Text.Comparator TEXT_COMPARATOR
= new Text.Comparator();
    public FirstComparator() {
        super(TextPair.class);
    }
    @Override
    public int compare(byte[] b1, int s1, int L1,
byte[] b2, int s2, int L2) {
        try {
```

```

decodeVIntSize(b1[s1]) +
decodeVIntSize(b2[s2]) +
compare(b1, s1, firstL1, b2,
ption(e);

```

```

ritableComparable a,
instanceof TextPair) {
areTo(((TextPair) b). first);

```

java object serialization?

Serialization mechanism, called referred to simply as "Java" is not used in Hadoop. Here's the answer to that question: Why we first started Hadoop? and I thought we needed we had precise control over read, since that is central to Hadoop. We can get some control, but

It was similar. Effective, communications are critical to precisely control how and buffers are handled, and those. The problem is that the criteria for a serialization mechanism, fast, extensible, and

compact: it writes the data to the stream this is

true of classes that implement java.io.Serializable or java.io.Externalizable. Subsequent instances of the same class write a reference handle to the first occurrence, which occupies only 5 bytes. However, reference handles don't work well with random access, since the referent class may occur at any point in the preceding stream that is, there is state stored in the stream. Even worse, reference handles play havoc with sorting records in a serialized stream, since the first record of a particular class is distinguished and must be treated as a special case.

All these problems are avoided by not writing the classname to the stream at all, which is the approach that Writable takes. This makes the assumption that the client knows the expected type. The result is that the format is considerably more compact than Java Serialization, and random access and sorting work as expected since each record is independent of the others (so there is no stream state).

Java Serialization is a general-purpose mechanism for serializing graphs of objects, so it necessarily has some overhead for serialization and deserialization operations. What's more, the deserialization procedure creates a new instance for each object deserialized from the stream. Writable objects, on the other hand, can be (and often are) reused. For example, for a MapReduce job, which at its core serializes and deserializes billions of records of just a handful of different types, the savings gained by not having to allocate new objects are significant.

In terms of extensibility, Java Serialization has some support for evolving a type, but it is brittle and hard to use effectively (Writables have no support: the programmer has to manage them himself). In principle, other languages could interpret the Java Serialization stream protocol (defined by the Java Object Serialization Specification), but in practice there are no widely used implementations in other languages, so it is a Java-only solution. The situation is the same for Writables.

Q.15 Why do we need WritableComparable?

OR

What happens if WritableComparable is not present?

Ans. We need to make our custom type, comparable if we want to compare this type with the other. We want to make our custom type as a key, then we should definitely make our

Big Data Analytics

key type as WritableComparable rather than simply Writable. This enables the custom type to be compared with other types and it is also sorted accordingly. Otherwise, the keys won't be compared with each other and they are just passed through the network.

If we have made our custom type Writable rather than WritableComparable our data won't be compared with other data types. There is no compulsion that our custom types need to be WritableComparable until unless it is a key. Because values don't need to be compared with each other as keys.

If our custom type is a key then we should have WritableComparable or else the data won't be sorted.

Q.16 How to make our custom type, WritableComparable?

Ans. We can make custom type a WritableComparable by following the method below:

```

public class add implements writableComparable {
    public int a;
    public int b;
    public add() {
        this.a=a;
        this.b=b;
    }
    public void write(DataOutput out) throws IOException {
        out.writeInt(a);
        out.writeInt(b);
    }
    public void readFields(DataInput in) throws
    IOException {
        a = in.readInt();
        b = in.readInt();
    }
    public int CompareTo(add c){
        int presentValue=this.value;
        int CompareValue=c.value;
        return (presentValue < CompareValue ? -1 :
        (presentValue == CompareValue ? 0 : 1));
    }
    public int hashCode() {

```

return In
IntToInt

}
}

These re
data faster in th

With
WritableCompa
custom type w
developers to
requirement.

**Q.17 Explain th
with Writ
implemen**

Ans. Serializati
structured objec
network or for w
is the reverse pr
series of structur
distinct areas of
communication
interprocess com
implemented usin
protocol uses seri
stream to be sent
the binary stream
RPC serialization

Compact : A con
bandwidth, which

Fast : Interproce
a distributed syst
performance over
deserialization pr

Extensible : Pre
requirements, so
protocol in a conti

Interoperable : F
to support clients
the server, so the
possible.

```
return Integer.parseInt(bits(a) ^ Integer.parseInt(bits(b));
```

These read fields and write make the comparison of data faster in the network.

With the use of these WritableComparable in Hadoop, we can make our serializable custom type with less difficulty. This gives the ease of requirement.

PART-C

Q17 Explain the significance of Writable interface along with Writable Comparable and comparators with implementing the serialization.

Ans. Serialization : Serialization is the process of turning structured objects into a byte stream for transmission over network or for writing to persistent storage. Deserialization is the reverse process of turning a byte stream back into a series of structured objects. Serialization appears in two distinct areas of distributed data processing: for interprocess communication and for persistent storage. In Hadoop, interprocess communication between nodes in the system is implemented using remote procedure calls (RPCs). The RPC protocol uses serialization to tender the message into a binary stream to be sent to the remote node, which then deserializes the binary stream into the original message. In general, RPC serialization format is:

Compact : A compact format makes the best use of network bandwidth, which is the most scarce resource in a data center.

Fast : Interprocess communication forms the backbone of a distributed system, so it is essential that there is as little performance overhead as possible for the serialization and deserialization process.

Extensible : Protocols change over time to meet new requirements, so it should be straightforward to evolve a protocol in a controlled manner for clients and servers.

Interoperable : For some systems, it is desirable to be able to support clients that are written in different languages on the server, so the format needs to be designed to make it possible.

BDA.46

Hadoop uses its own serialization format, Writables, which is certainly compact and fast, but not so easy to extend or use from languages other than Java.

The Writable Interface : The Writable interface defines two methods: one for writing its state to a DataOutput binary stream by using write(), and one for reading its state from a DataInput binary stream by using readFields() as below :

```
package org.apache.hadoop.io;
import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;
public interface Writable
{
    void write(DataOutput out) throws IOException;
    void readFields(DataInput in) throws IOException;
}
```

We will use IntWritable, a wrapper for a Java int. We can create one and set its value using the set() method:

```
IntWritable writable = new IntWritable();
writable.set(163);
```

Equivalently, we can use the constructor that takes the integer value:

```
IntWritable writable = new IntWritable(163);
```

To examine the serialized form of the IntWritable, we write a small helper method that wraps a java.io.ByteArrayOutputStream in a java.io.DataOutputStream to capture the bytes in the serialized stream:

```
public static byte[] serialize(IntWritable writable) throws IOException
{
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    DataOutputStream dataOut = new DataOutputStream(out);
    writable.write(dataOut);
    dataOut.close();
    return out.toByteArray();
}
```

```
An integer is written using four bytes
byte[] bytes = serialize(writable);
assertThat(bytes.length, is(4));
```

B.Tech. (VIII Sem.) C.S. Solved Papers

The bytes are written in big-endian order and we can see their hexadecimal representation by using a method on Hadoop's StringUtils:

```
assertThat(StringUtils.toHexString(bytes), is("000000a3"));
```

Let's try deserialization. Again, we create a helper method to read a Writable object from a byte array:

```
public static byte[] deserialize(Writable writable, byte[] bytes)
throws IOException
{
    ByteArrayInputStream in = new ByteArrayInputStream(bytes);
    DataInputStream dataIn = new DataInputStream(in);
    writable.readFields(dataIn);
    dataIn.close();
    return bytes;
}
```

We construct a new, value-less, IntWritable, then call deserialize() to read from the output data that we just wrote. Then we check that its value, retrieved using the get() method, is the original value, 163:

```
IntWritable newWritable = new IntWritable();
deserialize(newWritable, bytes);
assertThat(newWritable.get(), is(163));
```

WritableComparable and Comparators

IntWritable implements the WritableComparable interface, which is just a subinterface of the Writable and java.lang.Comparable interfaces:

```
package org.apache.hadoop.io;
public interface WritableComparable<T> extends Writable, Comparable<T>
{
}
```

Comparison of types is crucial for MapReduce, where there is a sorting phase during which keys are compared with one another. One optimization that Hadoop provides is the RawComparator extension of Java's Comparator:

```
package org.apache.hadoop.io;
import java.util.Comparator;
```

Q.20 Write detailed note on writable wrapper class.

Ans. Text : Text is a Writable for UTF-8 sequences. It can be thought of as the writable equivalent of `java.lang.String`. Text is a replacement for the UTF8 class, which was deprecated because it didn't support strings whose encoding was over 32,767 bytes, and because it used Java's modified UTF-8.

The Text class uses an `Int` (with a variable-length encoding) to store the number of bytes in the string encoding, so the maximum value is 2 GB. Furthermore, Text uses standard UTF-8, which makes it potentially easier to interoperate with other tools that understand UTF-8.

Indexing : Because of its emphasis on using standard UTF-8, there are some differences between Text and the Java String class. Indexing for the Text class is in terms of position in the encoded byte sequence, not the Unicode character in the string, or the Java char code unit (as it is for String). For ASCII strings, these three concepts of index position coincide. Here is an example to demonstrate the use of the `charAt()` method:

```
Text t = new Text("hadoop");
assertThat(t.getLength(), is(6));
assertThat(t.getBytes().length, is(6));
assertThat(t.charAt(2), is((int) 'd'));
assertThat("Out of bounds", t.charAt(100), is(-1));
```

Notice that `charAt()` returns an `int` representing a Unicode code point, unlike the String variant that returns a `char`. Text also has a `find()` method, which is analogous to String's `indexOf()`:

```
Text t = new Text("hadoop");
assertThat("Find a substring", t.find("do"), is(2));
assertThat("Finds first 'o'", t.find("o"), is(3));
assertThat("Finds 'o' from position 4 or later",
t.find("o", 4), is(4));
assertThat("No match", t.find("pig")), is(-1));
```

Unicode : When we start using characters that are encoded with more than a single byte, the differences between Text and String become clear. Consider the Unicode characters shown in Table.

Table : Unicode characters

Unicode code point	U+0041	U+00DF	U+6771	U+10400
Name	LATIN CAPITAL LETTER A	LATIN SMALL LETTER SHARP S	N/A (a unified Han ideograph)	DESERET CAPITAL LETTER LONG I
UTF-8 code units	41	c3 9f	e6 9d b1	f0 90 90 80
Java representation	\u0041	\u00DF	\u6771	\u10400

All but the last character in the table, U+10400, can be expressed using a single Java char. U+10400 is a supplementary character and is represented by two Java chars, known as a surrogate pair. The tests in Example 1 show the differences between String and Text when processing a string of the four characters from Table.

Example 1 : Tests showing the differences between the String and Text classes

```
public class StringTextComparisonTest {
    @Test
    public void string() throws UnsupportedOperationException {
        String s = "\u0041\u00DF\u6771\u10400";
        assertThat(s.length(), is(5));
        assertThat(s.getBytes("UTF-8").length, is(10));
        assertThat(s.indexOf("\u0041"), is(0));
        assertThat(s.indexOf("\u00DF"), is(1));
        assertThat(s.indexOf("\u6771"), is(2));
        assertThat(s.indexOf("\u10400"), is(3));

        assertThat(s.charAt(0), is('\u0041'));
        assertThat(s.charAt(1), is('\u00DF'));
        assertThat(s.charAt(2), is('\u6771'));
        assertThat(s.charAt(3), is('\u10400'));
        assertThat(s.charAt(4), is('\uDC00'));

        assertThat(s.codePointAt(0), is(0x0041));
        assertThat(s.codePointAt(1), is(0x00DF));
        assertThat(s.codePointAt(2), is(0x6771));
```

```
assertThat(codePointAt(3), is(0x10400));
```

```
@Test
```

```
public void test() {
```

```
Text t = new Text("u0041u00DFu6771uD801uDC00");
```

```
assertThat(t.getLength(), is(10));
```

```
assertThat(t.find("u0041"), is(0));
```

```
assertThat(t.find("u00DF"), is(1));
```

```
assertThat(t.find("u6771"), is(3));
```

```
assertThat(t.find("uD801uDC00"), is(6));
```

```
assertThat(t.charAt(0), is(0x0041));
```

```
assertThat(t.charAt(1), is(0x00DF));
```

```
assertThat(t.charAt(3), is(0x6771));
```

```
assertThat(t.charAt(6), is(0x10400));
```

```
}
```

The test confirms that the length of a String is the number of char code units it contains (5, one from each of the first three characters in the string, and a surrogate pair from the last), whereas the length of a Text object is the number of bytes in its UTF-8 encoding (10 = 1+2+3+4). Similarly, the `indexOf()` method in String returns an index in char code units, and `find()` for Text is a byte offset.

The `charAt()` method in String returns the char code unit for the given index, which in the case of a surrogate pair will not represent a whole Unicode character. The code `PointAt()` method, indexed by char code unit, is needed to retrieve a single Unicode character represented as an int. In fact, the `charAt()` method in Text is more like the `codePointAt()` method than its namesake in String. The only difference is that it is indexed by byte offset.

Iteration : Iterating over the Unicode characters in Text is complicated by the use of byte offsets for indexing, since you can't just increment the index. The idiom for iteration is

a little obscure (see Example 2): turn the Text object into a

Java.nio.ByteBuffer, then repeatedly call the `bytesToCodePoint()` static method on Text with the buffer.

This method extracts the next code point as an int and updates the position in the buffer. The end of the string is detected when `bytesToCodePoint()` returns -1.

Example 2 : Iterating over the characters in a Text object

```
public class TextIterator {
```

```
public static void main(String[] args) {
```

```
Text t = new Text("u0041u00DFu6771uD801uDC00");
```

```
ByteBuffer buf = ByteBuffer.wrap(t.getBytes(), 0,
```

```
t.getLength());
```

```
int cp;
```

```
while (buf.hasRemaining() && (cp =
```

```
Text.bytesToCodePoint(buf)) != -1) {
```

```
System.out.println(Integer.toHexString(cp));
```

```
}
```

```
}
```

Running the program prints the code points for the four characters in the string:

```
% hadoop TextIterator
```

```
41
```

```
df
```

```
6771
```

```
10400
```

Mutability : Another difference with String is that Text is mutable (like all writable implementations in Hadoop, except `NullWritable`, which is a singleton). You can reuse a Text instance by calling one of the `set()` methods on it. For example:

```
Text t = new Text("hadoop");
```

```
t.set("pig");
```

```
assertThat(t.getLength(), is(3));
```

```
assertThat(t.getBytes().length, is(3));
```

In some situations, the byte array returned by the `getBytes()` method may be longer than the length returned by `length()`:

```
Text t = new Text("hadoop");
```

```
t.set(new Text("pig"));
```

```
assertThat(t.getLength(), is(3));
```

```
assertThat("Byte length not shortened", t.getBytes().
```

```
length,
```

```
is(6));
```

This shows why it is imperative that you always call `getLength()` when calling `getBytes()`, so you know how much of the byte array is valid data.

Resorting to String : Text doesn't have as rich an API for manipulating strings as java.lang.String, so in many cases, you need to convert the Text object to a String. This is done in the usual way, using the `toString()` method:

```
assertThat(new Text("hadoop").toString(),
```

```
is("hadoop"));
```

BytesWritable

`BytesWritable` is a wrapper for an array of binary data. Its serialized format is an integer field (4 bytes) that specifies the number of bytes to follow, followed by the bytes themselves. For example, the byte array of length two with values 3 and 5 is serialized as a 4-byte integer (00000002) followed by the two bytes from the array (03 and 05):

```
BytesWritable b = new BytesWritable(new byte[] {3,
```

```
5});
```

```
byte[] bytes = serialize(b);
```

```
assertThat(StringUtils.byteToHexString(bytes),
```

```
is("000000020305"));
```

`BytesWritable` is mutable, and its value may be changed by calling its `set()` method. As with Text, the size of the byte array returned from the `getBytes()` method for `BytesWritable`—the capacity—may not reflect the actual size of the data stored in the `BytesWritable`. You can determine the size of the `BytesWritable` by calling `getLength()`. To demonstrate:

```
b.setCapacity(11);
```

```
assertThat(b.getLength(), is(2));
```

```
assertThat(b.getBytes().length, is(11));
```

NullWritable

`NullWritable` is a special type of Writable, as it has zero-length serialization. No bytes are written to, or read from the stream. It is used as a placeholder, for example, `MapReduce`, a key or a value can be declared a `NullWritable` when you don't need to use that position. `NullWritable` effectively stores a constant empty value. `NullWritable` also be useful as a key in `SequenceFile` when you want to store a list of values, as opposed to key-value pairs. It is an immutable singleton: the instance can be retrieved by calling `NullWritable.get()`.

ObjectWritable and GenericWritable

`ObjectWritable` is a general-purpose wrapper for the following: Java primitives, String, enum, Writable, and arrays of any of these types. It is used in Hadoop `RM` to marshal and unmarshal method arguments and return values.

`ObjectWritable` is useful when a field can be of a than one type: for example, if the values in a SequenceFile have multiple types, then you can declare the value type an `ObjectWritable` and wrap each type in an `ObjectWritable`. Being a general-purpose mechanism, it's fairly wasteful space since it writes the classname of the wrapped type each time it is serialized. In cases where the number of types is small and known ahead of time, this can be improved having a static array of types, and using the index into array as the serialized reference to the type. This is an approach that `GenericWritable` takes, and you have to subscribe to specify the types to support.

Q.21 Explain how to implement a custom variable?

Ans. Implementing a Custom Writable : Hadoop comes with a useful set of Writable implementations that serve purposes; however, on occasion, you may need to write your own custom implementation. With a custom Writable, you have full control over the binary representation and the order. Because Writables are at the heart of the MapReduce

Big Data Analytics

Example: Group_data = GROUP Relation_name BY AGE

The order statement is used to display the contents of relation in sorted order based on one or more fields.

Example: Relation_2 = ORDER Relation_name1 BY (ASC|DSC)

Distinct statement removes duplicate records and is implemented only on entire records, and not on individual records.

Example: Relation_2 = DISTINCT Relation_name1

Q.15 What are the relational operators in Pig?

Ans. The relational operators in Pig are as follows:

COGROUP : It joins two or more tables and then performs GROUP operation on the joined table result.

CROSS : This is used to compute the cross product (cartesian product) of two or more relations.

FOREACH : This will iterate through the tuples of a relation, generating a data transformation.

JOIN : This is used to join two or more tables in a relation.

LIMIT : This will limit the number of output tuples.

SPLIT : This will split the relation into two or more relations.

UNION : It will merge the contents of two relations.

ORDER : This is used to sort a relation based on one or more fields.

Q.16 Write a short note on admiring the Pig architecture.

Ans. Admiring the Pig Architecture : Pig is made up of two components:

(i) **The Language Itself :** The programming language for Pig is known as Pig Latin, a high-level language that allows you to write data processing and analysis programs.

(ii) **The Pig Latin Compiler :** The Pig Latin compiler converts the Pig Latin code into executable code. The executable code is either in the form of MapReduce jobs or it can spawn a process where a virtual Hadoop instance is created to run the Pig code on a single node.

The sequence of MapReduce programs enables Pig programs to do data processing and analysis in parallel, leveraging Hadoop MapReduce and HDFS. Running the Pig job in the virtual Hadoop instance is a useful strategy for

testing your Pig scripts. Figure shows how Pig relates to Hadoop ecosystem.

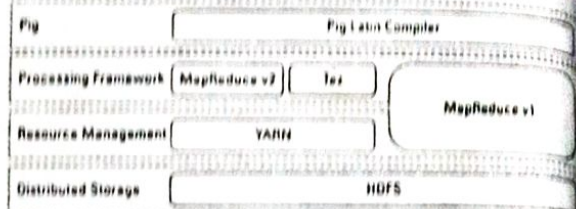


Fig. : Pig architecture

Pig programs can run on MapReduce 1 or MapReduce 2 without any code changes, regardless of what mode the cluster is running. However, Pig scripts can also run on the Tez API instead. Apache Tez provides a more efficient execution framework than MapReduce. YARN enables application frameworks other than MapReduce (like Tez) to run on Hadoop. Hive can also run against the Tez framework.

Q.17 Explain grunt.

Ans. Grunt : Grunt has line-editing facilities like those found in GNU Readline (used in the bash shell and many other command-line applications). For instance, the Ctrl-E combination will move the cursor to the end of the line. Grunt remembers command history, too, and you can recall lines from the history buffer using Ctrl-P or Ctrl-N (for previous or next) or, equivalently, the up or down cursor keys.

Another handy feature is Grunt's completion mechanism, which will try to complete Pig Latin keywords and functions when you press the Tab key. For example, consider the following incomplete line:

```
grunt> a = foreach b ge
```

If you press the Tab key at this point, ge will expand to generate, a Pig Latin keyword:

```
grunt> a = foreach b generate
```

You can customize the completion tokens by creating a file named *auto complete* and placing it on Pig's classpath (such as in the conf directory in Pig's install directory) or in the directory you invoked Grunt from. The file should contain one token per line, and tokens must not contain any whitespace. Matching is case-sensitive. It can be very useful to add commonly used file paths (especially because Pig does not perform file-name completion) or the names of any user-defined functions you have created.

You can get a list of commands using the help command. When you've finished your Grunt session, you can exit with the quit command.

Q.18 Why do we need Apache Pig?

Ans. Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

- Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Q.19 Write the features of Pig.

Ans. Features of Pig : Apache Pig comes with the following features :

1. **Rich set of operators :** It provides many operators to perform operations like join, sort, filter, etc.
2. **Easy of programming :** Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
3. **Optimization opportunities :** The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
4. **Extensibility :** Using the existing operators, users can develop their own functions to read, process, and write data.
5. **UDF's :** Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.

6. **Handles all kinds of data :** Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Q.20 Write the differences between Apache Pig and MapReduce.

Ans. Apache Pig Vs MapReduce : Listed below are the major differences between Apache Pig and MapReduce.

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

Q.21 Explain Pig Latin application flow.

Ans. At its core, Pig Latin is a dataflow language, where you define a data stream and a series of transformations that are applied to the data as it flows through your application. This is in contrast to a control flow language (like C or Java), where you write a series of instructions.

In control flow languages, you use constructs like loops and conditional logic (like an if statement). You won't find loops and if statements in Pig Latin.

If you need some convincing that working with Pig is a significantly easier row to hoe than having to write Map

and Reduce programs, start by taking a look at some real Pig syntax:

```
A = LOAD 'data_file.txt';
```

```
B = GROUP ...;
```

```
C = FILTER ...;
```

```
DUMP B;
```

```
STORE C INTO 'Results';
```

Some of the text in this example actually looks like English, right? Not too scary, at least at this point. Looking at each line in turn, you can see the basic flow of a Pig program. (Note that this code can either be part of a script or issued on the interactive shell called Grunt.)

1. Load: You first load (LOAD) the data you want to manipulate. As in a typical MapReduce job, that data is stored in HDFS. For a Pig program to access the data, you first tell Pig what file or files to use. For that task, you use the LOAD 'data_file' command.

Here, 'data_file' can specify either an HDFS file or a directory. If a directory is specified, all files in that directory are loaded into the program.

If the data is stored in a file format that isn't natively accessible to Pig, you can optionally add the USING function to the LOAD statement to specify a user-defined function that can read in (and interpret) the data.

2. Transform: You run the data through a set of transformations that, way under the hood and far removed from anything you have to concern yourself with, are translated into a set of Map and Reduce tasks.

The transformation logic is where all the data manipulation happens. Here, you can FILTER out rows that aren't of interest, JOIN two sets of data files, GROUP data to build aggregations, ORDER results, and do much, much more.

3. Dump: Finally, you dump (DUMP) the results to the screen or Store (STORE) the results in a file somewhere.

You would typically use the DUMP command to send the output to the screen when you debug your programs. When your program goes into production, you simply change the DUMP call to a STORE call so that any results from

running
process

Q.22

Ans.
data
data,
Pig's
Data

LO

DU

ST

FO

FIL

GR

JO

Tra

C

C

D

U

S

C

Q.23 What are the advantages of using Pig over MapReduce?

Ans. In Mapreduce :

- (i) Development cycle is very long. Writing mappers and reducers, compiling.
- (ii) Packaging the code, submitting jobs, and retrieving the results is a time.
- (iii) Consuming process.
- (iv) Performing Data set joins is very difficult.
- (v) Low level and rigid, and leads to a great deal of custom user code that is hard to maintain and reuse is complex.

In pig :

- (i) No need of compiling or packaging of code. Pig operators will be converted into map or reduce tasks internally.
- (ii) Pig Latin provides all of the standard data-processing operations, such as join, filter, group by, order by, union, etc.
- (iii) High level of abstraction for processing large data sets.

Q.24 Explain Pig script interfaces in Hadoop.

Ans. The Pig programming language is designed to handle any kind of data tossed its way – structured, semi structured, unstructured data, you name it. Pig programs can be packaged in three different ways:

1. **Script:** This method is nothing more than a file containing Pig Latin commands, identified by the .pig suffix (FlightData.pig, for example). Ending your Pig program with the .pig extension is a convention but not required. The commands are interpreted by the Pig Latin compiler and executed in the order determined by the Pig optimizer.
2. **Grunt:** Grunt acts as a command interpreter where you can interactively enter Pig Latin at the Grunt command line and immediately see the response. This method is helpful for prototyping during initial development and with what-if scenarios.
3. **Embedded:** Pig Latin statements can be executed within Java, Python, or JavaScript programs.

Pig scripts, Grunt shell Pig commands, and embedded Pig programs can run in either Local mode or MapReduce mode.

The Grunt shell provides an interactive shell to submit Pig commands or run Pig scripts. To start the Grunt shell in Interactive mode, just submit the command pig at your shell.

To specify whether a script or Grunt shell is executed locally or in Hadoop mode just specify it in the -x flag to the pig command. The following is an example of how you'd specify running your Pig script in local mode:

```
 pig -x local milesPerCarrier.pig
```

Here's how you'd run the pig script in Hadoop mode, which is the default if you don't specify the flag:

```
 pig -x mapreduce milesPerCarrier.pig
```

[Note : By default, when you specify the Pig command without any parameters, it starts the Grunt shell in Hadoop mode. If you want to start the Grunt shell in local mode just add the -x local flag to the command. Here is an example : pig -x local]

Q.25 Explain nested model and also explain interactive modes of Pig.

Ans. Pig Latin has a fully-nestable data model with Atomic values, Tuples, Bags or lists, and Maps. This implies one data type can be nested within another. Pig Latin Nested Data Model is shown in the following Fig.1.

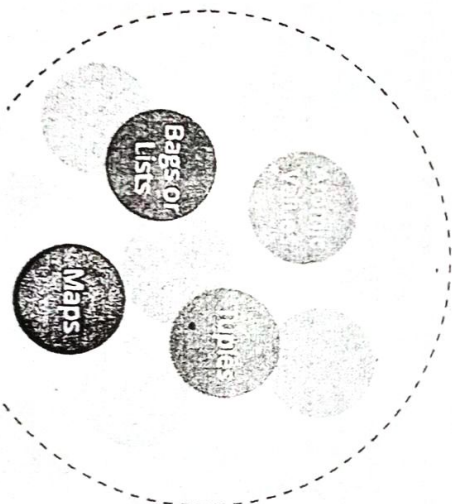


Fig. 1

The advantage is that this is more natural to programmers than flat Tuples. Also, it avoids expensive joins.

Now we will look into different execution modes pig works in.

Interactive Mode : Interactive mode means coding and executing the script, line by line, as shown in the image given below.

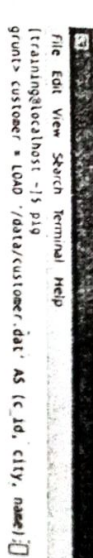


Fig. 2

Batch Mode : In Batch mode, all scripts are coded in a file with the extension .pig and the file is directly executed as shown in the diagram given below :

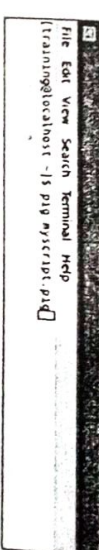


Fig. 3

PART-C

Q.26 Write detailed note on Pig.

Ans. Pig raises the level of abstraction for processing large datasets. MapReduce allows you the programmer to specify a map function followed by a reduce function, but working out how to fit your data processing into this pattern, which often requires multiple MapReduce stages, can be a challenge. With Pig, the data structures are much richer, typically being multivalued and nested, and the set of transformations you can apply to the data are much more powerful—they include joins, for example, which are not for the faint of heart in MapReduce.

Pig is made up of two pieces:

- The language used to express data flows, called Pig Latin.
- The execution environment to run Pig Latin programs. There are currently two environments: Local execution in a single JVM and distributed execution on a Hadoop cluster.

A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input

data to produce a result. Under the hood, MapReduce is unaware of it than the native.

Pig is a very long, and packaging the results is screaming, which the experience to process terabytes of data.

lines of Pig Latin code to make the huge data programmer write commands for in program, as it is a sample run on a real you can see whether you can see whether

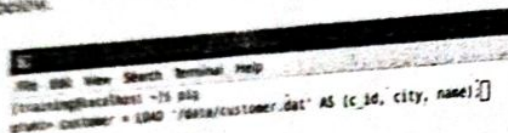
Pig was designed of the processing pipeline, grouping, and functions (UDFs). The data model, so they operators. As another, Pig isn't suitable Like MapReduce, it is a If you want to perform amount of data in a large well, since it is set up large portions of it.

In some cases, programs written in narrowing with each sophisticated algorithm operators. It's fair to a lot of effort optimizing queries in Pig Latin

Big Data Analytics

Now we will look into different execution modes pig works in.

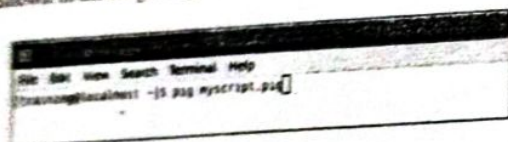
Interactive Mode : Interactive mode means coding and executing the script, line by line, as shown in the image given below.



```
gr> load '/data/customer.dat' AS (c_id, city, name);
```

Fig. 2

Batch Mode : In Batch mode, all scripts are coded in a file with the extension .pig and the file is directly executed as shown in the diagram given below :



```
gr> pig myscript.pig
```

Fig. 3

PART-C

Q.36 Write detailed note on Pig.

Ans. Pig raises the level of abstraction for processing large datasets. MapReduce allows you the programmer to specify a map function followed by a reduce function, but working out how to fit your data processing into this pattern, which often requires multiple MapReduce stages, can be a challenge. With Pig, the data structures are much richer, typically being multidimensional and nested; and the set of transformations you can apply to the data are much more powerful—they include joins, for example, which are not for the faint of heart in MapReduce.

Pig is made up of two pieces:

- The language used to express data flows, called Pig Latin.
- The execution environment to run Pig Latin programs. There are currently two environments: Local execution in a single JVM and distributed execution on a Hadoop cluster.

A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input

data to produce output. Taken as a whole, the operations describe a data flow, which the Pig execution environment translates into an executable representation and then runs. Under the covers, Pig turns the transformations into a series of MapReduce jobs, but as a programmer you are mostly unaware of this, which allows you to focus on the data rather than the nature of the execution.

Pig is a scripting language for exploring large datasets. One criticism of MapReduce is that the development cycle is very long. Writing the mappers and reducers, compiling and packaging the code, submitting the job(s), and retrieving the results is a time-consuming business, and even with screaming, which removes the compile and package step, the experience is still involved. Pig's sweet spot is its ability to process terabytes of data simply by issuing a half-dozen lines of Pig Latin from the console. Indeed, it was created at Yahoo! to make it easier for researchers and engineers to mine the huge datasets there. Pig is very supportive of a programmer writing a query, since it provides several commands for introspecting the data structures in your program, as it is written. Even more useful, it can perform a sample run on a representative subset of your input data, so you can see whether there are errors in the processing before unleashing it on the full dataset.

Pig was designed to be extensible. Virtually all parts of the processing path are customizable: Loading, storing, filtering, grouping, and joining can all be altered by user-defined functions (UDFs). These functions operate on Pig's nested data model, so they can integrate very deeply with Pig's operators. As another benefit, UDFs tend to be more reusable than the libraries developed for writing MapReduce programs.

Pig isn't suitable for all data processing tasks, however. Like MapReduce, it is designed for batch processing of data. If you want to perform a query that touches only a small amount of data in a large dataset, then Pig will not perform well, since it is set up to scan the whole dataset, or at least large portions of it.

In some cases, Pig doesn't perform as well as programs written in MapReduce. However, the gap is narrowing with each release, as the Pig team implements sophisticated algorithms for implementing Pig's relational operators. It's fair to say that unless you are willing to invest a lot of effort optimizing Java MapReduce code, writing queries in Pig Latin will save you time.

APPLYING STRUCTURE TO HADOOP DATA WITH HIVE

IMPORTANT QUESTIONS

PART-A

Q.1 What is Hive?

Ans. Hive : Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy. Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Q.2 What is the difference between an external table and a managed table in Hive?

Ans. Difference between an external table and a managed table

External Table	Managed Table
External tables in Hive refer to the data that is at an existing location outside the warehouse directory	Also known as the internal table, these types of tables manage the data and move it into its warehouse directory by default
Hive deletes the metadata information of a table and does not change the table data present in HDFS	If one drops a managed table, the metadata information along with the table data is deleted from the Hive warehouse directory

Q.3 What are the features of hive?

Ans. Features of Hive

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Q.4 What is a partition in Hive?

Ans. Partition in Hive : Partition is a process for grouping similar types of data together based on columns or partition keys. Each table can have one or more partition keys to identify a particular partition.

Q.5 Why is partitioning required in Hive?

Ans. Partitioning provides granularity in a Hive table. It reduces the query latency by scanning only relevant partitioned data instead of the entire data set. We can partition the transaction data for a bank based on month – January, February, etc. Any operation regarding a particular month, say February, will only have to scan the February partition, rather than the entire table data.

Q.6 Why does Hive not store metadata information in HDFS?

Ans. We know that the Hive's data is stored in HDFS. However, the metadata is either stored locally or it is stored

Big Data Analytics

in RDBMS. The metadata is not stored in HDFS, because HDFS read/write operations are time-consuming. As such, Hive stores metadata information in the metastore using RDBMS instead of HDFS. This allows us to achieve low latency and is faster.



Fig.

Q.7 What are the components used in Hive query processors?

Ans. The components used in Hive query processors are:

- Parser
- Semantic Analyzer
- Execution Engine
- User-Defined Functions
- Logical Plan Generation
- Physical Plan Generation
- Optimizer
- Operators
- Type checking

PART-B

Q.8 Give a introduction to hive (hello to Hive).

Ans. Hive provides Hadoop with a bridge to the RDBMS world and provides an SQL dialect known as Hive Query Language (HiveQL), which can be used to perform SQL-like tasks. That's the big news, but there's more to Hive than meets the eye, as they say, or more applications of this new technology than you can present in a standard elevator pitch. For example, Hive also makes possible the concept known as enterprise data warehouse (EDW) augmentation, a leading use case for Apache Hadoop, where data warehouses are set up as RDBMSs built specifically for data analysis and reporting. Now, some experts will argue that Hadoop (with Hive, HBase, Sqoop, and its assorted buddies) can replace

the EDW, but we disagree. We believe that Apache Hadoop is a great addition to the enterprise and that it can augment and complement existing EDWs.

Closely associated with RDBMS/EDW technology is extract, transform, and load (ETL) technology. To grasp what ETL does, it helps to know that, in many use cases, data cannot be immediately loaded into the relational database – it must first be extracted from its native source, transformed into an appropriate format, and then loaded into the RDBMS or EDW. For example, a company or an organization might extract unstructured text data from an Internet forum, transform the data into a structured format that's both valuable and useful, and then load the structured data into its EDW.

Hive is a powerful ETL tool in its own right, along with the major player in this realm: Apache Pig. Again, users may try to set up Hive and Pig as the new ETL tools for the data center. (Let them try.) As with the debate over Last but not least, Apache Hive gives you powerful analytical tools, all within the framework of HiveQL. These tools should look and feel quite familiar to IT professionals who understand how to use SQL.

Q.9 What are the key differences between Hive and Pig?

Ans.

Hive	Pig
It uses a declarative language, called HiveQL, which is similar to SQL for reporting.	Uses a high-level procedural language called Pig Latin for programming
Operates on the server-side of the cluster and allows structured data.	Operates on the Client side of the cluster and allows both structured and unstructured data
It does not support the Avro file format by default. This can be done using "Org.Apache.Hadoop.Hive.serde2.Avro"	Supports Avro file format by default.
Facebook developed it and it supports partition	Yahoo developed it and it does not support partition

Q.1) Explain HIVE CLI client.

USER INTERFACES

WEB UI

MVCE COMMAND LINE

HD Input



Unit Name	Operation
-----------	-----------

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (in Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadatabase of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Ans. The Hive CLI client : To master the finer points of the Hive CLI client, it might help to revisit the (somewhat busy-looking) Hive architecture diagram shown in Figure, we've streamlined the original figure to focus only on the components that are required when running the CLI.



to run the Hive CLI, you execute the Hive command `hive` and specify the CLI as the service you want to run. In Listing 1, you can see the command that's required as well as some of our first HiveQL statements. (We have included a steps annotation using the A-B-C model in the listing to direct your attention to the key commands.)

Listing 1: Using the Hive CLI to Create a Table

- ```
(A) $HIVE_HOME/bin hive --service cli
(B) hive > set hive.cli.print.current.db=true;
(C) hive (default) > CREATE DATABASE
ourfirstdatabase;
OK
Time taken: 3.756 seconds
(D) hive (default) > USE ourfirstdatabase;
```

OK

(E) hive

```
(E) hive (ourfirstdatabase) > CREATE TABLE
our_first_cable
```

—

```
> FirstName
```

LASTNAME  
SIRI

OK

**BDA**

Time taken: 0.043 seconds  
hive (ourfirstdatabase) > quit;

```
(F) $ls/home/biadmin/live/warehouse/ourfirstdatabase.db
our_first_table
```

The first command in Listing 1 (see Step A) starts the Hive CLI using the `$HIVE_HOME` environment variable:

**in .bashrc**

```
export HADOOP_HOME=/home/user/Hive/hadoop
hadoop-1.2.1
```

```
export JAVA_HOME=/opt/jdk
```

```
export HIVE_HOME=/home/user/Hive/hive-0.11.0
```

```
export PATH=$HADOOP_HOME/
```

bin:\$HIVE\_HOME/bin:

```
$JAVA_HOME/bin: $PATH
```

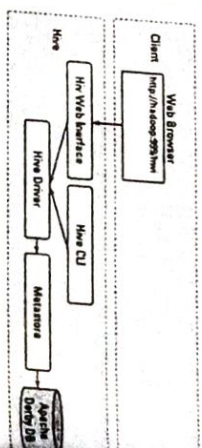
The service `cli` command-line option directs the Hive system to start the command-line interface, though you could have chosen other servers. (In fact, you can try a few later in this section.) Next, in Step B, you tell the Hive CLI to print your current working database so that you know where you are in the namespace. (This statement will make sense after we explain how to use the next command, so hold tight.) Continuing in Listing 1, in Step C you use HiveQL's data definition language (DDL) to create your first database.

**Ans. The Web Browser as Hive client :** Using the CLI requires only one command to start the Hive shell, when you want to access Hive using a web browser, first need to start the HWJ Server and then point your browser to the port on which the server is listening. Figure 1 illustrates how this type of Hive client configuration might work. (Note that even though you might not be using the Hive CLI, not an optional component and is still present.)

**Q.12** Write a note on web browser as HIVE client.

**Ans. The Web Browser as Hive client : Using the**

CLL requires only one command to start the Hive shell, when you want to access Hive using a web browser, first need to start the HWJ Server and then point your browser to the port on which the server is listening. Figure 1 illustrates how this type of Hive client configuration might work. (Note that even though you might not be using the Hive CLI, not an optional component and is still present.)



**Fig. 1 : The Hive Web Interface Client Configuration**

before you can start the HWI Server:

1. Using the commands in Listing 1 (following this page), you can create the `hive-site.xml` file and configure the `$HIVE_HOME/conf/hive-site.xml` to ensure that Hive can find and load the HWT's server pages.
2. The HWT Server requires Apache Ant libraries N

You set the `hive.metastore.warehouse.dir` variable to point to the local directory `/home/biadmin/Hive/warehouse` in your Linux virtual machine rather than use the HDFS as you would on a proper Hadoop cluster.

so you need to download more files. Download from the Apache site at <http://ant.apache.org/bindownload.cgi>.

1

**Compression:** Data compression cannot only save space in HDFS but also improve performance by reducing the overall size of input/output operations. Additionally, compression between the Hadoop mappers and reducers can improve performance, because less data is passed between nodes in the cluster. Hive supports intermediate compression between the mappers and reducers as well as table output compression. Hive also understands how to ingest compressed data into the warehouse. Files compressed with Gzip or Bzip2 can be read by Hive's LOAD DATA command.

**Functions:** HiveQL provides a rich set of built-in operators, built-in functions, built-in aggregate functions, and built-in-table-generating functions. To list all built-in functions for any particular Hive release, use the SHOW FUNCTIONS HiveQL command. You can also retrieve information about a built-in function by using the HiveQL commands DESCRIBE FUNCTION EXTENDED function\_name and DESCRIBE FUNCTION EXTENDED function\_name. Using the EXTENDED keyword sometimes returns usage examples for the specified built-in function. Additionally, Hive allows users to create their own functions, called user-defined functions, or UDFs. Using Hive's Java-based UDF framework, you can create additional functions, including aggregates and table-generating functions. This feature is one of the reasons that Hive can function as an ETL tool.

#### Q.15 Explain various types of joins in HIVE QL.

**Ans. JOIN :** There are different types of joins given as follows:

- JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

The following query executes JOIN on the CUSTOMER and ORDER tables and retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN
ORDERS o ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response

| ID | NAME     | AGE | AMOUNT |
|----|----------|-----|--------|
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | khilan   | 25  | 1560   |
| 4  | chaitali | 25  | 2060   |

**LEFT OUTER JOIN :** The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right table, the JOIN still returns a row in the result, but with NULL in each column from the right table. A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL, in case of no matching JOIN predicate.

The following query demonstrates LEFT OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c LEFT
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

| ID | NAME     | AMOUNT | DATE                |
|----|----------|--------|---------------------|
| 1  | Ramesh   | 1560   | 2009-11-20 00:00:00 |
| 2  | khilan   | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | chaitali | 2060   | 2008-05-20 00:00:00 |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |

**RIGHT OUTER JOIN :** The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

The following query demonstrates RIGHT OUTER JOIN between the CUSTOMER and ORDER tables.

#### Big Data Analytics

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c RIGHT
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following resp. use:

| ID | NAME     | AMOUNT | DATE       |
|----|----------|--------|------------|
| 3  | kaushik  | 3000   | 2009-10-08 |
| 3  | kaushik  | 1500   | 2009-11-20 |
| 2  | khilan   | 1560   | 2008-05-20 |
| 4  | chaitali | 2060   | 2008-05-20 |

#### FULL OUTER JOIN

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfill the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c FULL
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

| ID | NAME     | AMOUNT | DATE                |
|----|----------|--------|---------------------|
| 1  | Ramesh   | 1560   | 2009-11-20 00:00:00 |
| 2  | khilan   | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2  | khilan   | 1560   | 2009-11-20 00:00:00 |
| 4  | chaitali | 2060   | 2008-05-20 00:00:00 |

#### PART-C

#### Q.16 Explain the working of HIVE with neat sketch.

**Ans.** The following diagram depicts the workflow between Hive and Hadoop

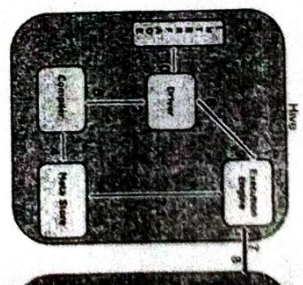


Fig.

The followings define how H framework:

1. **Execute Query :** The Command Line or Web UI: database driver such as JDBC
2. **Get Plan :** The driver takes that parses the query to create plan or the requirement of
3. **Get Metadata :** The compiler to Metastore (any database response to the compiler:
4. **Send Metadata :** Metastore response to the compiler:
5. **Send Plan :** The compiler and resends the plan to the parsing and compiling of a
6. **Execute Plan :** The driver the execution engine.
7. **Execute Job :** Internally job is a MapReduce job. The job is a JobTracker, which assigns this job to TaskTracker. Here, the query executes
8. **Metadata Ops :** Metadata execution engine can interact with Metastore.
9. **Fetch Result :** The execution results from Data nodes.
10. **Send Results to the Driver :** sends those resultant values to the results to Hive Interf

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c RIGHT
```

```
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following result set:

| ID | NAME     | AMOUNT | DATE       |
|----|----------|--------|------------|
| 3  | kaushik  | 3000   | 2009-10-08 |
| 3  | kaushik  | 1500   | 2009-10-08 |
| 2  | khilan   | 1560   | 2009-11-20 |
| 4  | Chaitali | 2060   | 2008-05-20 |

### FULL OUTER JOIN

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c FULL
```

```
OUTER JOIN ORDERS o ON (c.ID =
o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

| ID | NAME     | AMOUNT | DATE                |
|----|----------|--------|---------------------|
| 1  | Ramesh   | NULL   | NULL                |
| 2  | khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | hardik   | NULL   | NULL                |
| 6  | sonu     | NULL   | NULL                |
| 7  | ruffy    | NULL   | NULL                |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2  | khilan   | 1560   | 2009-11-20 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |

### PART-C

**Q.16** Explain the working of HIVE with neat sketch.

**Ans.** The following diagram depicts the workflow between Hive and Hadoop

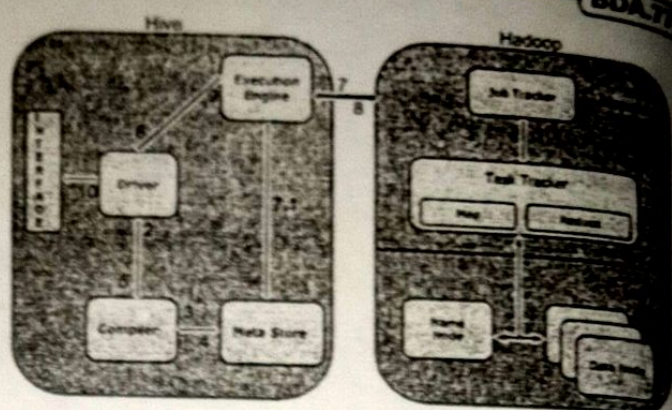


Fig.

The followings define how Hive interacts with Hadoop framework:

1. **Execute Query** : The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2. **Get Plan** : The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3. **Get Metadata** : The compiler sends metadata request to Metastore (any database).
4. **Send Metadata** : Metastore sends metadata as a response to the compiler.
5. **Send Plan** : The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6. **Execute Plan** : The driver sends the execute plan to the execution engine.
7. **Execute Job** : Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
  - **Metadata Ops** : Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8. **Fetch Result** : The execution engine receives the results from Data nodes.
9. **Send Results to the Driver** : The execution engine sends those resultant values to the driver.
10. **Send Results to Hive Interface** : The driver sends the results to Hive Interfaces.

a types:

Syntax: ARRAY&lt;data\_type&gt;

Maps : Maps in Hive are similar to Java Maps.

Syntax : MAP&lt;primitive\_type: data\_type&gt;

Structs : Structs in Hive is similar to using complex data with comment.

Syntax: STRUCT&lt;col\_name: data\_type [COMMENT col\_comment], ...&gt;

amp with  
imestamp  
nd format

month'day

me as Big  
immutable  
follows:neous data  
union. The

, struct'a:

othing but  
of data isit floating  
type. The  
to 10308.

he special

es are as

they are

Q.19 Write detailed note on creating and managing database and tables.

**Ans. Hive - Create Database :** Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it. Hive contains a default database named default.

#### Create Database Statement

- Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows: CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
- Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named userdb:

```
hive > CREATE DATABASE [IF NOT EXISTS]
userdb;
```

or

```
hive > CREATE SCHEMA userdb;
```

- The following query is used to verify a databases list:

```
hive> SHOW DATABASES;
```

default

userdb

#### Drop Database Statement

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

```
DROP DATABASE Statement DROP
(DATABASE|SCHEMA) [IF EXISTS]
database_name [RESTRICT | CASCADE];
```

The following queries are used to drop a database. Let us assume that the database name is userdb.

```
hive> DROP DATABASE IF EXISTS userdb;
```

The following query drops the database using CASCADE. It means dropping respective tables before dropping the database.

```
hive> DROP DATABASE IF EXISTS userdb
CASCADE;
```

The following query drops the database using SCHEMA.

```
hive> DROP SCHEMA userdb;
```

#### Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT
EXISTS] [db_name.] table_name
```

```
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
```

```
[ROW FORMAT row_format] [STORED AS file_format]
```

Example

Let us assume you need to create a table named employee using CREATE TABLE statement. The following table lists the fields and their data types in employee table:

| S.No. | Field Name  | Data Type |
|-------|-------------|-----------|
| 1.    | Eid         | int       |
| 2.    | Name        | String    |
| 3.    | Salary      | Float     |
| 4.    | Designation | string    |

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

```
COMMENT Employee details FIELDS
TERMINATED BY \n LINES TERMINATED
BY \n STORED IN TEXT FILE
```

The following query creates a table named employee using the above data.

```
hive> CREATE TABLE IF NOT EXISTS
employee(eid int, name String, salary String, destination
String)
```

```
COMMENT Employee details FIELDS
TERMINATED BY \n LINES TERMINATED
BY \n STORED AS
```

If you add the option IF NOT EXISTS in the statement in case the table already exists.

On successful creation of the following response:

OK

Time taken; 5.905 seconds

hive&gt;

#### Load Data Statement

Generally, after creating a table, you can load data using the Insert statement. But you can also load data using the LOAD DATA statement.

While inserting data into Hive, you can use LOCAL to store bulk records. There are two types of data: one is from local file system and the other is from HDFS.

While inserting data into Hive, you can use LOCAL to store bulk records. There are two types of data: one is from local file system and the other is from HDFS.

Syntax

The syntax for load data is as follows:

```
LOAD DATA [LOCAL] IN
[OVERWRITE] INTO TABLE tablename
[partcol1=val1, partcol2=val2 ...]
```

- LOCAL is identifier to specify that the data is from local file system.
- OVERWRITE is optional to overwrite the table.
- PARTITION is optional.

#### Example

We will insert the following data into a text file named sample.txt in /home/user directory.

|      |             |       |             |
|------|-------------|-------|-------------|
| 1201 | Gopal       | 45000 | Technician  |
| 1202 | Manisha     | 45000 | Proofreader |
| 1203 | Masthanvali | 40000 | Technician  |
| 1204 | Kiran       | 40000 | Hr Admin    |
| 1205 | Kranthi     | 30000 | Op Admin    |

1 to drop a database.  
is userdb.

IF EXISTS userdb  
the database using  
rective tables before

the database using

ed to create a table in  
follows:

col\_\_comment[, ...])  
[AL] TABLE [IF NOT  
RED AS file\_\_format]

create a table named  
ntment. The following  
es in employee table:

| Data Type |
|-----------|
| int       |
| String    |
| Float     |
| string    |

ant, Row formatted fields  
ninator, and Stored File

details FIELDS  
MINATED

LE  
a table named employee

E IF NOT. EXISTS  
alary String, destination

B.Tech. (VII Sem.) C.S. Solved Papers

COMMENT Employee details ROW FORMAT  
DELIMITED FIELDS TERMINATED BY \n  
TERMINATED BY \n STORED AS TEXTFILE;

If you add the option IF NOT EXISTS, Hive ignores

the statement in case the table already exists

On successful creation of table, you get to see the  
following response:

OK

Time taken: 5.905 seconds

hive>

Load Data Statement

Generally, after creating a table in SQL, we can insert  
data using the Insert statement. But in Hive we can insert  
data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD  
DATA to store bulk records. There are two ways to load  
data: one is from local file system and second is from Hadoop  
file system

While inserting data into Hive, it is better to use LOAD  
DATA to store bulk records. There are two ways to load data:  
one is from local file system and second is from Hadoop  
file system

Syntax

The syntax for load data is as follows:

LOAD DATA [LOCAL] INPATH 'filepath'  
[OVERWRITE] INTO TABLE tablename [PARTITION  
[partcol1=val1, partcol2=val2 ...]]

- LOCAL is identifier to specify the local path. It is  
optional
- OVERWRITE is optional to overwrite the data in the  
table.
- PARTITION is optional.

Example

We will insert the following data into the table. It is a  
text file named sample.txt in /home/user directory.

|      |             |       |                   |
|------|-------------|-------|-------------------|
| 1201 | Gopal       | 45000 | Technical manager |
| 1202 | Manisha     | 45000 | Proof reader      |
| 1203 | Masthanvali | 40000 | Technical writer  |
| 1204 | Kiran       | 40000 | Hr Admin          |
| 1205 | Kranthi     | 30000 | Op Admin          |

Big Data Analytics

The following query loads the given text into the table

hive> LOAD DATA LOCAL INPATH '/home/user/  
sample.txt' OVERWRITE INTO TABLE employee;

On successful download, you get to see the following  
response:

OK

Time taken : 15.905 seconds

hive>

Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on  
what attributes we wish to modify in a table.

ALTER TABLE name RENAME TO new\_name

ALTER TABLE name ADD COLUMNS (col\_spec[,  
col\_spec ...])

ALTER TABLE name DROP [COLUMN] column\_name

ALTER TABLE name CHANGE column\_name new\_name  
new\_type

ALTER TABLE name REPLACE COLUMNS (col\_spec[,  
col\_spec...])

Rename To Statement

The following query renames the table from employee  
to emp.

hive> ALTER TABLE employee RENAME TO emp;

Change Statement

The following table contains the fields of employee  
table and it shows the fields to be changed (in bold).

| Field       | Convert from | Change field | Convert to |
|-------------|--------------|--------------|------------|
| name        | data type    | name         | data type  |
| eid         | int          | eid          | int        |
| name        | String       | ename        | String     |
| salary      | Float        | salary       | Double     |
| designation | String       | designation  | String     |

The following queries rename the column name and  
column data type using the above data:

hive> ALTER TABLE employee CHANGE name

ename String;

hive> ALTER TABLE employee CHANGE salary

salary Double;

BDA87

Add Columns Statement

The following query adds a column named dept to the  
employee table.

hive> ALTER TABLE employee ADD COLUMNS  
(dept STRING COMMENT 'Department name');

Replace Statement

The following query deletes all the columns from the  
employee table and replaces it with emp and name columns:

hive> ALTER TABLE employee REPLACE  
COLUMNS (eid INT empid Int, ename STRING name  
String);

Drop Table Statement

Hive Metastore, it removes the table/column data and  
their metadata. It can be a normal table (stored in Metastore)  
or an external table (stored in local file system). Hive treats  
both in the same manner, irrespective of their types.

The syntax is as follows:

DROP TABLE [IF EXISTS] table\_name;

The following query drops a table named employee:

hive> DROP TABLE IF EXISTS employee;

On successful execution of the query, you get to  
response:

OK

hive>

The following query is used to verify the list of tables:

hive> SHOW TABLES;

emp ok

Time taken: 2.1 seconds

hive>

Creating, Dropping and Altering Databases in Apache

Hive is as below -

- (1) \$ SHIVE\_HOME/bin hive --service cli
- (2) hive> set hive.cli.print.current.db = true;
- (3) hive (default)> USE ourfirstdatabase;
- (4) hive (ourfirstdatabase)> ALTER DATABASE  
ourfirstdatabase SET DBPROPERTIES

('creator'='Bruce Brown',

'created\_for'='Learning Hive DDL');

OK

Time taken: 0.138 seconds

```
(i) Hive> CREATE TABLE data_types_table(
```

**INPUTFORMAT, OUTPUTFORMAT:** INPUTFORMAT will read data from the Hive table, OUTPUTFORMAT does the same thing for writing data to the Hive table. To see the default settings for the table, simply execute a DESCRIBE

BDAs

for building structured data in Hive tables from unstructured logs, semi-structured log files, e-mails, tweets, and other data from social media. Regular expressions allow us to extract meaningful information.

our smallint smallint 2 byte

(Origin='JFK' AND Dest='ORD')  
hive> SELECT \* FROM myFlightInfo2008;  
OK

|      |   |      |      |      |     |     |
|------|---|------|------|------|-----|-----|
| 2008 | 7 | 910  | 1103 | 5199 | JFK | ORD |
| 2008 | 7 | 705  | 849  | 5687 | JFK | ORD |
| 2008 | 7 | 1645 | 1914 | 5469 | JFK | ORD |
| 2008 | 7 | 1345 | 1514 | 4392 | JFK | ORD |
| 2008 | 7 | 1718 | 1907 | 1217 | JFK | ORD |
| 2008 | 7 | 757  | 929  | 1323 | JFK | ORD |
| 2008 | 7 | 928  | 1057 | 907  | JFK | ORD |
| 2008 | 7 | 1358 | 1532 | 915  | JFK | ORD |
| 2008 | 7 | 1646 | 1846 | 917  | JFK | ORD |
| 2008 | 7 | 2129 | 2341 | 919  | JFK | ORD |

Time taken: 0.186 seconds, Fetched: 10 row(s)

#### Listing: An Example of Using CREATE TABLE AS SELECT

In Step A, we build two smaller tables derived from the FlightInfo2007 and FlightInfo2008 by selecting a subset of fields from the larger tables for a particular day (in this case, July 3), where the origin of the flight is New York's JFK airport (JFK) and the destination is Chicago's O'Hare airport (ORD). Then in Step B we simply dump the contents of these small tables so that you can view the data.

#### Q.21 Explain querying and analyzing the data.

**Ans. Joining tables with Hive:** Well, remember that the underlying operating system for Hive is (surprise!) Apache Hadoop: MapReduce is the engine for joining tables, and the Hadoop File System (HDFS) is the underlying storage. Disk and network access is a lot slower than memory access, so minimize HDFS reads and writes as much as possible. Hive table reads and writes via HDFS usually involve very large blocks of data, the more data you can manage altogether in one table, the better the overall performance.

Now we show you a Hive join example using our flight data tables. The above Listing shows you how to create and display a myFlightInfo2007 table and a myFlightInfo2008 table from the larger FlightInfo2007 and FlightInfo2008 tables. The plan all along was to use the CTAS created myFlightInfo2007 and myFlightInfo2008 tables to illustrate how you can perform joins in Hive. The plan all along was to use the CTAS created

myFlightInfo2007 and myFlightInfo2008 tables to illustrate how you can perform joins in Hive. Figure 1 shows the result of an inner join with the myFlightInfo2007 and myFlightInfo2008 tables using the Squirrel SQL client.

Fig. 1: The Hive Inner Join

Hive supports equi-joins, a specific type of join that only uses equality comparisons in the join predicate. Other comparators such as Less Than (<) are not supported. This restriction is only because of limitations on the underlying MapReduce engine. Also, you cannot use OR in the ON clause.

Figure 2 illustrates how an inner join works using a Venn diagram technique. The basic idea here is that an inner join returns the records that match between two tables. So an inner join is a perfect analysis tool to determine which flights are the same from JFK (New York) to ORD (Chicago) in July of 2007 and July of 2008.

#### Hive Join Examples



Fig. 3: Hive inner join, full outer join, and left outer join

Improving your Hive queries with indexes: Creating an index is common practice with relational databases when

we want to speed access to a column or set of columns in your database. Without an index, the database system has to read all rows in the table to find the data we have selected. Indexes become even more essential when the tables grow extremely large. Hive supports index creation on tables. In below Listing, we list the steps necessary to index the FlightInfo2008 table.

(A) CREATE INDEX f08\_index ON TABLE FlightInfo2008

(Origin) AS 'COMPACT' WITH DEFERRED REBUILD;

(B) ALTER INDEX f08\_index ON FlightInfo2008 REBUILD;

(C) hive (flightdata)> SHOW INDEXES ON FlightInfo2008;

OK

f08index FlightInfo2008 origin  
flightdata\_\_FlightInfo2008\_\_f08index\_\_compact  
Time taken: 0.079 seconds, Fetched: 1 row(s)

(D) hive (flightdata)> DESCRIBE

flightdata\_\_FlightInfo2008\_\_f08index\_\_;

OK

origin string None  
\_bucketname string  
\_offsets array<bigint>

(E) Time taken: 0.112 seconds, Fetched: 3 row(s)

hive (flightdata)> SELECT Origin, COUNT(\*) FROM FlightInfo2008 WHERE Origin = 'SYR' GROUP BY Origin;

SYR 12032

(F) Time taken: 17.34 seconds, Fetched: 1 row(s)

hive (flightdata)> SELECT Origin, SIZE(\_offsets) FROM flightdata\_\_flightinfo2008\_\_f08index\_\_ WHERE origin = 'SYR';

SYR 12032

Time taken: 8.347 seconds, Fetched: 1 row(s)

(G) hive (flightdata)> DESCRIBE

flightdata\_\_FlightInfo2008\_\_f08index\_\_;

OK

origin string None  
\_bucketname string  
\_offsets array<bigint>

Time taken: 0.12 seconds, Fetched: 3 row(s)

#### Listing: Creating an Index on the FlightInfo2008 Table

Step (A) creates the index using the 'COMPACT' index handler on the Origin column. Hive also offers a bitmap index handler as of the 0.8 release, which is intended for creating indexes on columns with a few unique values. The keywords WITH DEFERRED REBUILD instructs Hive to first create an empty index. Step (B) is where we actually build the index with the ALTER INDEX REBUILD command. Deferred index builds can be very useful in workflows where one process creates the tables and indexes, another loads the data and builds the indexes and a final process performs data analysis. Hive doesn't provide automatic index maintenance, so you need to rebuild the index if you overwrite or append data to the table. Also, Hive indexes support table partitions, so a rebuild can be limited to a partition. Step (C) illustrates how we can list or show the indexes created against a particular table. Step (D) illustrates an important point regarding

**Hive Indexes:** Hive indexes are implemented as tables. This is why we need to first create the index table and then build it to populate the table. Therefore, we can use indexes in at least two ways:

- Count on the system to automatically use indexes that you create.
- Rewrite some queries to leverage the new index table.

In Step (E) we write a query that seeks to determine how many flights left the Syracuse airport during 2008.

To get this information, we leverage the COUNT aggregate function. In Step (F), you leverage the new index table and use the SIZE function instead.

**Windowing in HiveQL:** The concept of windowing, introduced in the SQL:2003 standard, allows the SQL programmer to create a frame from the data against which aggregate and other window functions can operate. HiveQL now supports windowing per the SQL standard. One question we had when we first discovered this data set was, "What exactly is the average flight delay per day?" So we created a query in below Listing that produces the average departure delay per day in 2008.

(A) hive (flightdata)> CREATE VIEW avgdepdelay AS  
> SELECT DayOfWeek, AVG(DepDelay) FROM  
FlightInfo2008 GROUP BY DayOfWeek;  
OK

Time taken: 0.121 seconds

(B) hive (flightdata)> SELECT \* FROM avgdepdelay;

OK

1. 10.269990244459473
2. 8.97689712068735
3. 8.289761053658728
4. 9.772897177836702
5. 12.158036387869656
6. 8.645680904903614
7. 11.568973392595312

Time taken: 18.6 seconds, Fetched: 7 row(s)

#### Listing : Finding the Average Departure Delay per Day in 2008

As shown in step(A) of above Listing Hive's Data Definition Language (DDL) also includes the CREATE VIEW statement, which can be quite useful. In Hive, views allow a query to be saved but data is not stored as with the Create Table as Select (CTAS) statement.

Suppose if you want to know "What is the first flight between Airport X and Y?" Suppose that in addition to this information, you want to know about subsequent flights. just in case you're not a "morning person." Write the query as below.

(A) hive (flightdata)> SELECT f08.Month,  
f08.DayOfMonth

cr.description, f08.Origin, f08.Dest,

f08.FlightNum, f08.DepTime, MIN(f08.DepTime)

OVER (PARTITION BY f08.DayOfMonth ORDER  
BY f08.DepTime)

FROM flightinfo2008 f08 JOIN Carriers cr ON

f08.UniqueCarrier = cr.code

WHERE f08.origin = 'JFK' AND f08.Dest = 'ORD'

AND

f08.Month = 1 AND f08.DepTime != 0

...

OK

|   |   |                        |     |     |      |      |     |
|---|---|------------------------|-----|-----|------|------|-----|
| 1 | 1 | JetBlue Airways        | JFK | ORD | 903  | 641  | 641 |
| 1 | 1 | American Airlines Inc. | JFK | ORD | 1323 | 833  | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 907  | 929  | 641 |
| 1 | 1 | Comair Inc.            | JFK | ORD | 5083 | 945  | 641 |
| 1 | 1 | Comair Inc.            | JFK | ORD | 5634 | 1215 | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 915  | 1352 | 641 |
| 1 | 1 | American Airlines Inc. | JFK | ORD | 1323 | 833  | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 907  | 929  | 641 |
| 1 | 1 | Comair Inc.            | JFK | ORD | 5083 | 945  | 641 |
| 1 | 1 | Comair Inc.            | JFK | ORD | 5634 | 1215 | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 915  | 1352 | 641 |
| 1 | 1 | American Airlines Inc. | JFK | ORD | 1815 | 1610 | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 917  | 1735 | 641 |
| 1 | 1 | Comair Inc.            | JFK | ORD | 5469 | 1749 | 641 |
| 1 | 1 | Comair Inc.            | JFK | ORD | 5492 | 2000 | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 919  | 2102 | 641 |
| 1 | 1 | JetBlue Airways        | JFK | ORD | 919  | 48   | 48  |

#### Listing : Using Aggregate Window Functions on the Flight Data

In Step (A), we've replaced the GROUP BY clause with the OVER clause where we specify the PARTITION or window over which we want the MIN aggregate function to operate. We've also included the ORDER BY clause so that we can see those subsequent flights after the first one.

□□□