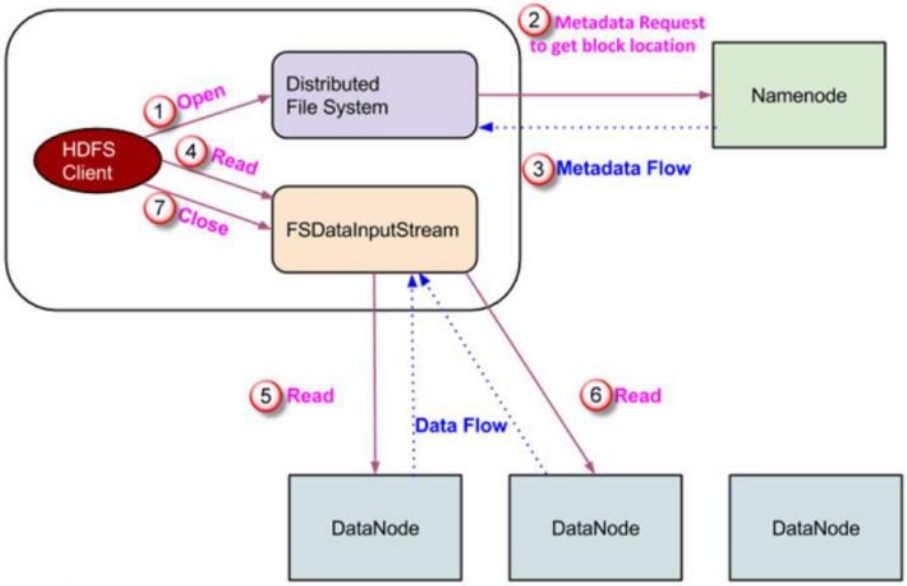
CMR INSTIT UTE OF TECHNLOGY	USN										

Internal AssessmentTest - 3

Sub:	Big Data Analytics					Sub Code:	17CS82	Branch:	ISE	
Date:	16/05/2019	Duration:	90 min's	Max Marks:	50	Sem/Sec:	VIII / A ,B			OBE

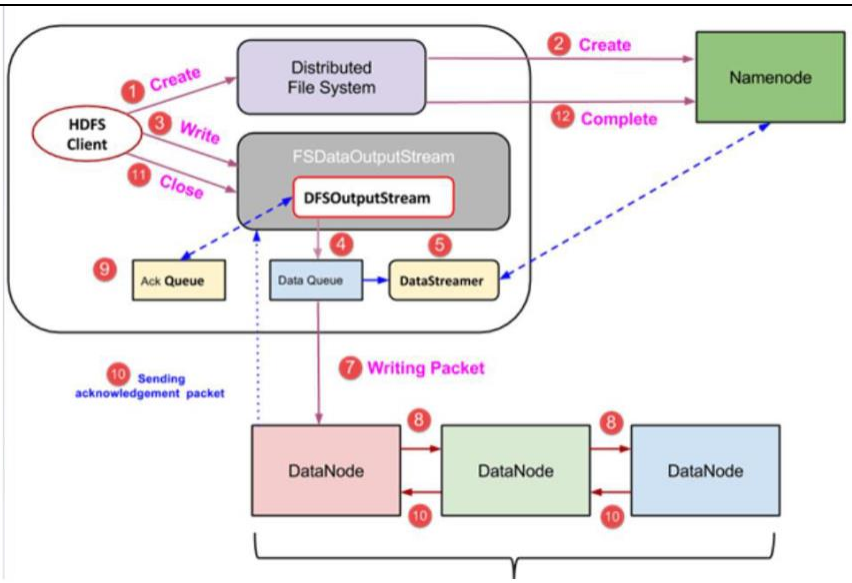
Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	**Explain the benefits of Big data processing and features of Hadoop.** **Solution:** **Benefits of Big Data Processing** Ability to process Big Data brings in multiple benefits, such as- • **Businesses can utilize outside intelligence while taking decisions** Access to social data from search engines and sites like facebook, twitter are enabling organizations to fine tune their business strategies. • **Improved customer service** Traditional customer feedback systems are getting replaced by new systems designed with Big Data technologies. In these new systems, Big Data and natural language processing technologies are being used to read and evaluate consumer responses. • **Early identification of risk to the product/services, if any** • **Better operational efficiency** **Features of Hadoop** • **Suitable for Big Data Analysis** As Big Data tends to be distributed and unstructured in nature, HADOOP clusters are best suited for analysis of Big Data. Since it is processing logic (not the actual data) that flows to the computing nodes, less network bandwidth is consumed. This concept is called as **data locality concept** which helps increase the efficiency of Hadoop based applications. **• Scalability** HADOOP clusters can easily be scaled to any extent by adding additional cluster nodes and thus allows for the growth of Big Data. Also, scaling does not require modifications to application logic. **• Fault Tolerance** HADOOP ecosystem has a provision to replicate the input data on to other cluster nodes. That way, in the event of a cluster node failure, data processing can still proceed by using data stored on another cluster node.	[10]	CO1	L2
2	**Explain HDFS Read and write operations in detail.** **Solution:** HDFS is a distributed file system for storing very large data files, running on clusters of commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. Hadoop comes bundled with **HDFS** (**Hadoop Distributed File Systems**). When data exceeds the capacity of storage on a single physical machine, it becomes essential to divide it across a number of separate machines. A file system that manages storage specific operations across a network of machines is called a distributed file system. HDFS is one such software. **Read Operation In HDFS** Data read request is served by HDFS, NameNode, and DataNode. Let's call the reader as a 'client'. Below diagram depicts file read operation in Hadoop.	[10]	CO1	L3

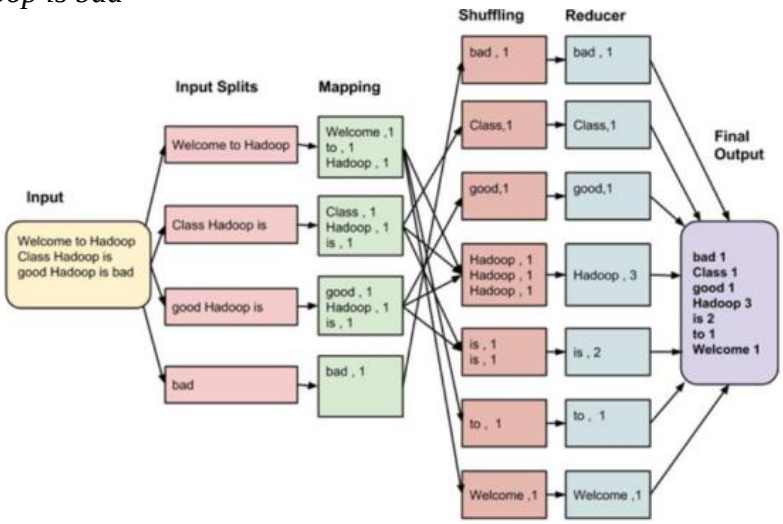


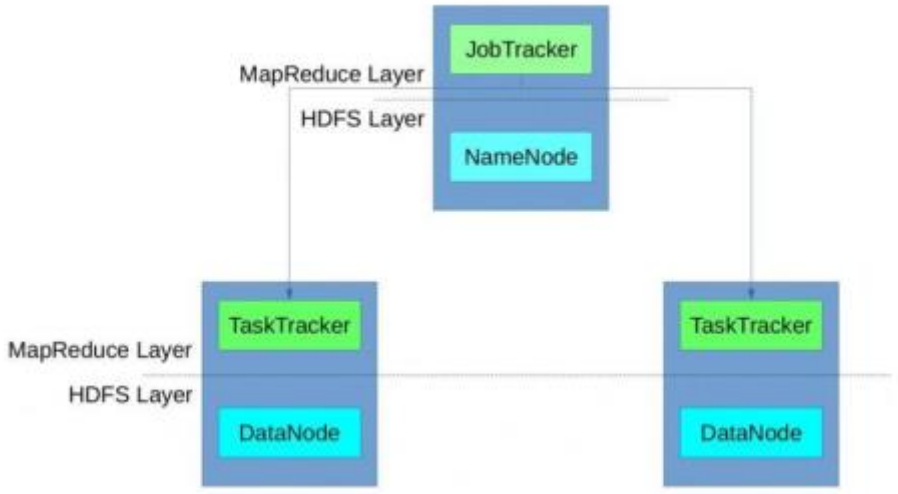
1.A client initiates read request by calling '**open()**' method of FileSystem object; it is an object of type **DistributedFileSystem**.
2. This object connects to namenode using RPC and gets metadata information such as the locations of the blocks of the file. Please note that these addresses are of first few blocks of a file.
3. In response to this metadata request, addresses of the DataNodes having a copy of that block is returned back.
4. Once addresses of DataNodes are received, an object of type **FSDataInputStream** is returned to the client. **FSDataInputStream** contains **DFSInputStream** which takes care of interactions with DataNode and NameNode. In step 4 shown in the above diagram, a client invokes '**read()**' method which causes **DFSInputStream** to establish a connection with the first DataNode with the first block of a file.
5. Data is read in the form of streams wherein client invokes '**read()**' method repeatedly. This process of **read()** operation continues till it reaches the end of block.
6. Once the end of a block is reached, DFSInputStream closes the connection and moves on to locate the next DataNode for the next block
7. Once a client has done with the reading, it calls **a close()** method.

Write Operation In HDFS
Lets understand how data is written into HDFS through files.



1. A client initiates write operation by calling 'create()' method of DistributedFileSystem object which creates a new file - Step no. 1 in the above diagram.
2. DistributedFileSystem object connects to the NameNode using RPC call and initiates new file creation. However, this file creates operation does not associate any blocks with the file. It is the responsibility of NameNode to verify that the file (which is being created) does not exist alreadyand a client has correct permissions to create a new file. If a file already exists or client does nothave sufficient permission to create a new file, then **IOException** is thrown to the client. Otherwise, the operation succeeds and a new record for the file is created by the NameNode.
3. Once a new record in NameNode is created, an object of type FSDataOutputStream is returned to the client. A client uses it to write data into the HDFS. Data write method is invoked (step 3 in the diagram).
4. FSDataOutputStream contains DFSOutputStream object which looks after communication withDataNodes and NameNode. While the client continues writing data, **DFSOutputStream** continues creating packets with this data. These packets are enqueuedinto a queue which is called as **DataQueue**.
5. There is one more component called **DataStreamer** which consumes this **DataQueue**. DataStreamer also asks NameNode for allocation of new blocks thereby picking desirable DataNodes to be used for replication.
6. Now, the process of replication starts by creating a pipeline using DataNodes. In our case, wehave chosen a replication level of 3 and hence there are 3 DataNodes in the pipeline.
7. The DataStreamer pours packets into the first DataNode in the pipeline.
8. Every DataNode in a pipeline stores packet received by it and forwards the same to the
second DataNode in a pipeline.
9. Another queue, 'Ack Queue' is maintained by DFSOutputStream to store packets which are waiting for acknowledgment from DataNodes.
10. Once acknowledgment for a packet in the queue is received from all DataNodes in the pipeline, it is removed from the 'Ack Queue'. In the event of any DataNode failure, packets from this queueare used to reinitiate the operation.
11. After a client is done with the writing data, it calls a close() method (Step 9 in the diagram) Call toclose(), results into flushing remaining data packets to the pipeline followed by waiting for acknowledgment.
12. Once a final acknowledgment is received, NameNode is contacted to tell it that

	the file write operation is complete.			
3	**What is Map Reduce? Explain how it works with example.** **Solution:** **What is MapReduce in Hadoop?** MapReduce is a programming model suitable for processing of huge data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster. **MapReduce programs work in two phases:** 1. Map phase 2. Reduce phase. An input to each phase is **key-value** pairs. In addition, every programmer needs to specify two functions: **map function** and **reduce function**. **How MapReduce Works:-** The whole process goes through four phases of execution namely, splitting, mapping,shuffling, and reducing. Let's understand this with an example – Consider you have following input data for your Map Reduce Program *Welcome to Hadoop Class* *Hadoop is good* *Hadoop is bad*  The final output of the MapReduce task is bad 1 Class 1 good 1 Hadoop 3 is 2 to 1 Welcome 1 The data goes through the following phases	[10]	CO1	L3

	Input Splits: An input to a MapReduce job is divided into fixed-size pieces called **input splits** Input split is a chunk of the input that is consumed by a single map **Mapping** This is the very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, a job of mapping phase is to count a number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency> **Shuffling** This phase consumes the output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, the same words are clubed together along with their respective frequency. **Reducing** In this phase, output values from the Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset. In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each word.			
4	**Discuss YARN architecture in detail.** **Solution:** **YARN Architecture** YARN (Yet Another Resource Negotiator) has been introduced to Hadoop with version 2.0 and solves a few issues with the resources scheduling of MapReduce in version 1.0. In order to understand the benefits of YARN, we have to review how resource scheduling worked in version 1.0. A MapReduce job is split by the framework into tasks (Map tasks, Reducer tasks) and each task is run on of the DataNode machines on the cluster. For the execution of tasks, each DataNode machine provided a predefined number of slots (map slots, reducers slots). The JobTracker was responsible for the reservation of execution slots for the different tasks of a job and monitored their execution. If the execution failed, it reserved another slot and re-started the task. It also cleaned up temporary resources and make the reserved slot available to other tasks.  The fact that there was only one JobTracker instance in Hadoop 1.0 led to the problem that the whole MapReduce execution could fail, if the the JobTracker fails (single point of failure). Beyond that, having only one instance of the JobTracker limits scalability (for very large clusters with thousands of nodes).	[10]	CO1	L2

	The concept of predefined map and reduce slots also caused resource problems in case all map slots are used while reduce slots are still available and vice versa. In general it was not possible to reuse the MapReduce infrastructure for other types of computation like real-time jobs. While MapReduce is a batch framework, applications that want to process large data sets stored in HDFS and immediately inform the user about results cannot be implemented with it. Beneath the fact that MapReduce 1.0 did not offer realtime provision of computation results, all other types of applications that want to perform computations on the HDFS data had to be implemented as Map and Reduce jobs, which was not always possible. Hence Hadoop 2.0 introduced YARN as resource manager, which no longer uses slots to manage resources. Instead nodes have "resources" (like memory and CPU cores) which can be allocated by applications on a per request basis. This way MapReduce jobs can run together with non-MapReduce jobs in the same cluster. The heart of YARN is the Resource Manager (RM) which runs on the master node and acts as a global resource scheduler. It also arbitrates resources between competing applications. In contrast to the Resource Manager, the Node Managers (NM) run on slave nodes and communicates with the RM. The NodeManager is responsible for creating containers in which the applications run, monitors their CPU and memory usage and reports them to the RM. Each application has its own ApplicationMaster (AM) which runs within a container and negotiates resources with the RM and works with the NM to execute and monitor tasks. The MapReduce implementation of Hadoop 2.0 therefore ships with an AM (named MRAppMaster) that requests containers for the execution of the map tasks from the RM, receives the container IDs from the RM and then executes the map tasks within the provided containers. Once the map tasks have finished, it requests new containers for the execution of the reduce tasks and starts their execution on the provided containers. If the execution of a task fails, it is restarted by the ApplicationMaster. Should the ApplicationMaster fail, the RM will attempt to the restart the whole application (up to two times per default). Therefore the ApplicationMaster can signal if it supports job recovery. In this case the ApplicationMaster receives the previous state from the RM and can only restart incomplete tasks. If a NodeManager fails, i.e the RM does not receive any heartbeats from it, it is removed from the list of active nodes and all its tasks are treated as failed. In contrast toversion 1.0 of Hadoop, the ResourceManager can be configured for High Availability.			
5	**List and Explain any five essential Hadoop tools with their features.** **Solution: Essential Hadoop Tools** Hadoop is an open source distributed processing framework which is at the center of a growing big data ecosystem. Used to support advanced analytics initiatives, including predictive analytics, data mining and machine learning applications, Hadoop manages data processing and storage for big data applications and can handle various forms of structured and unstructured data. **1. Hadoop Distributed File System** The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute	[10]	CO2	L1

user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. We describe the architecture of HDFS and report on experience using HDFS to manage 40 petabytes of enterprise data at Yahoo.

Features:

a. Rack awareness allows consideration of a node's physical location, when allocating storage and scheduling tasks

b. Minimal data motion. MapReduce moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides. This significantly reduces the network I/O patterns and keeps most of the I/O on the local disk or within the same rack and provides very high aggregate read/write bandwidth.

c. Utilities diagnose the health of the files system and can rebalance the data on different nodes

d. Rollback allows system operators to bring back the previous version of HDFS after an upgrade, in case of human or system errors

e. Standby NameNode provides redundancy and supports high availability

f. Highly operable. Hadoop handles different types of cluster that might otherwise require operator intervention. This design allows a single operator to maintain a cluster of 1000s f nodes.

2. Hbase

HBase is a column-oriented database management system that runs on top of HDFS. It is well suited for sparse data sets, which are common in many big data use cases. Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java much like a typical MapReduce application. HBase does support writing applications in Avro, REST, and Thrift.

Features:

a. Linear and modular scalability.

b. Strictly consistent reads and writes.

c. Automatic and configurable sharding of tables

d. Automatic failover support between RegionServers.

e. Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.

f. Easy to use Java API for client access.

g. Block cache and Bloom Filters for real-time queries.

h. Query predicate push down via server side Filters

3. HIVE

The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX.

Features:

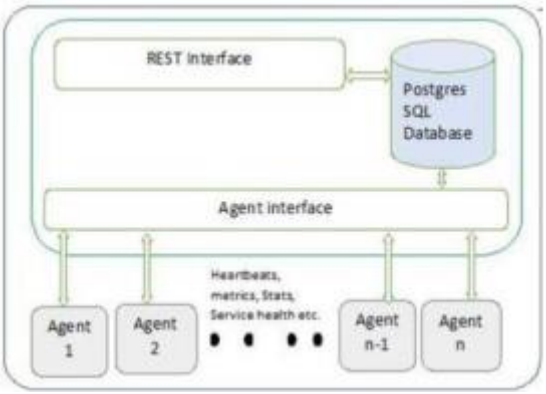
a. Indexing to provide acceleration, index type including compaction and Bitmap index as of 0.10, more index types are planned.

b. Different storage types such as plain text, RCFile, HBase, ORC, and others.

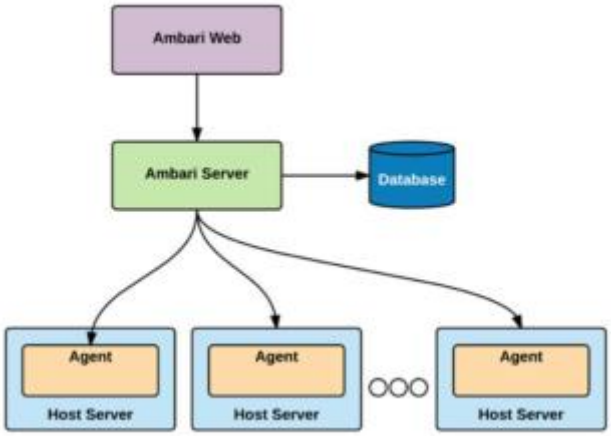
c. Metadata storage in an RDBMS, significantly reducing the time to perform semantic checks during query execution.

d. Operating on compressed data stored into Hadoop ecosystem, algorithm including gzip, bzip2, snappy, etc.

	e. Built-in user defined functions (UDFs) to manipulate dates, strings, and other data-mining tools. Hive supports extending the UDF set to handle use-cases not supported by built-in functions. f. SQL-like queries (Hive QL), which are implicitly converted into map-reduce jobs. **4. Sqoop** Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS. **Features:** a. Connecting to database server b. Controlling parallelism c. Controlling the import process d. Import data to hive e. Import data to Hbase **5. Pig** Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets. At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin **Features:** a. Ease of programming. b. It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain. c. Optimization opportunities. d. The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency. e. Extensibility. Users can create their own functions to do special-purpose processing.			
6	**Explain architecture of Apache Ambari.** **Solution:** **Apache Ambari architecture** Ambari provides intuitive and REST APIs that automate the operations in the Hadoop cluster. Its consistent and secure interface allows it to be fairly efficient in operational control. Its easy and user-friendly interface efficiently diagnoses the health of Hadoop cluster using an interactive dashboard.	[10]	CO2	L3



To have a better understanding of how Ambari works, let's look at the detailed architecture of Ambari, in the following diagram:



Apache Ambari follows a master/slave architecture where the master node instructs the slave nodes to perform certain actions and report back the state of every action. The master node is responsible for keeping track of the state of the infrastructure. To do this, the master node uses a database server, which can be configured during setup time.

These are the following applications in Apache Ambari, at the core:

• Ambari server
• The Ambari agent
• Ambari web UI
• Database

1. Ambari server

The entry point for all administrative activities on the master server is known as Ambari server. It is a shell script. Internally this script uses Python code, **ambari-server.py** and routes all the requests to it. Ambari server consists of several entry points that are available when passed different parameters to the Ambari-server program like:

• Daemon management
• Software upgrade
• Software setup
• LDAP (Lightweight Direct Access Protocol) /PAM (Pluggable Authentication Module)
/Kerberos management
• Ambari backup and restore
• Miscellaneous options

2. Ambari Agent

	The Ambari Agent runs on all the nodes that we want to manage with Ambari. This program periodically heartbeats to the master node. By using this agent, Ambari-server executes many of the tasks on the servers. **3. Ambari web interface** Ambari web interface is one of the powerful features of Ambari application. The web application is through the server of Ambari program which is running on the master host exposed on port 8080. You can access this application and this application is protected by authentication. Also, you can control and view all aspects of your Hadoop Cluster, once you log in to the web portal. **4. Database** Ambari supports multiple RDBMS (Relational Database Management Systems) to keep track of the state of the entire Hadoop infrastructure. Also, you can choose the database you want to use during the setup of the Ambari for the first time. Ambari supports these following databases at the time of writing: • PostgreSQL • Oracle • MySQL or MariaDB • Embedded PostgreSQL • Microsoft SQL Server • SQL Anywhere • Berkeley DB This technology is preferred by the big data developers as it is quite handy and comes with a step-by-step guide allowing easy installation on the Hadoop cluster. Its pre-configured key operational metrics provide quick look into the health of Hadoop core, i.e., HDFS and MapReduce along with the additional components such as Hive, HBase, HCatalog, etc. Ambari sets up a centralized security system by incorporating Kerberos and Apache Ranger into the architecture. The RESTful APIs monitor the information as well as integrate the operational tools. Its user-friendliness and interactivity has brought it in the range of top ten open source technologies for Hadoop cluster.			
7	**Discuss the features and benefits of Apache Ambari.** **Solution:** **Features of Apache Ambari** Following are some of features of Ambari. Read on to understand how the tool is expertly used in big data arena. **1. Platform independent –** Apache Ambari runs in Windows, Mac and many other platforms as it architecturally supports any hardware and software systems. Other platforms where Ambari runs are Ubuntu, SLES, RHEL etc. Those components which are dependent on a platform like yum, rpm packages, debian packages ought to be plugged with well defined interfaces. **2.Pluggable component –** Any current Ambari application can be customized. Any specific tools and technologies ought to be encapsulated by pluggable components. The goal of pluggability doesn't encompass standardization of inter-component. **3.Version management and upgrade –** Ambari itself maintains versions and hence there is no need of external tools like Git. If any Ambari application is to be upgraded or even Ambari is to be upgraded then doing it fairly easy. **4.Extensibility –** We can extend the functionality of existing Ambari applications by adding different view components. **5.Failure recovery –** Assume you are working on an Ambari application and something wrong happens. Then the system should gracefully recover from it. If you are a Windows user you can relate well to this. You might have worked on word file and suddenly there is a power outage. After turning the system on there	[10]	CO2	L2

will be an auto saved version of the document when you run the MS word.
6.Security – The Ambari application comes with robust security and it can sync with LDAP over the active directory.

Benefits of using Apache Ambari

This is given with respect to Hortonworks Data Platform (HDP). Ambari eliminates the need for manual tasks used to watch over Hadoop operations. It gives a simple secure platform for provisioning, managing and monitoring HDP deployments. Ambari is an easy to use Hadoop management UI and is solidly backed by REST APIs.

It provides numerous benefits like:

1.Installation, configuration and management is way simplified

Ambari can efficiently create Hadoop clusters at scale. It wizard driven approach lets the configuration be automated as per the environment so that the performance is optimal. Master slave and client components are assigned to configuring services. It is also used to install, start and test the cluster. Configuration blueprints give recommendations to those seeking a hands-on approach. The blue print of an ideal cluster is stored. How it is provisioned is clearly traced. This is then used to automate the creation of successive clusters without any user interaction. Blueprints also preserve and ensure the application of best practices across different environments. Ambari also provides rolling upgrade feature where running clusters can be updated on the go with maintenance releases and feature bearing releases and therefore there is no unnecessary downtime. When there are large clusters involved then rolling updates are simply not possible in which case express updates are used. Here the downtime is there but is minimum as when the update is manual. Both rolling and express updates are free of manual updates.

2. Centralized security and application

The complexity of cluster security configuration and administration is greatly reduced by Ambari which is among the components of Hadoop ecosystem. The tool also helps with automated setup of advanced security constructs like Kerboros and Ranger.

3. Complete visibility to cluster health

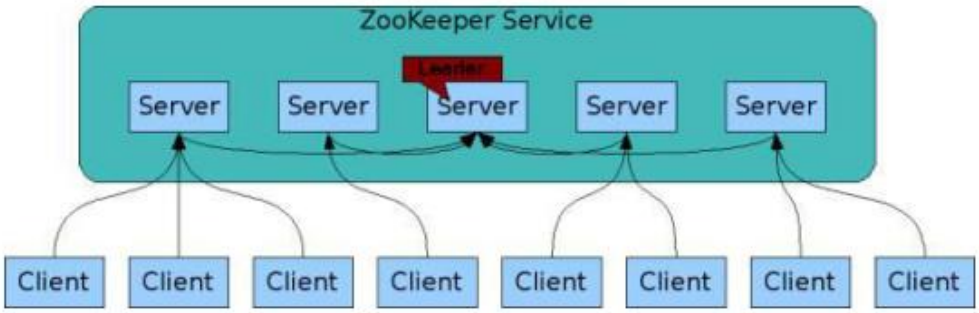
Through this tool you can monitor your cluster's health and availability. An easily customized web based dashboard has metrics that give status information for each service in the cluster like HDFS, YARN and HBase. The tool also helps with garnering and visualizing critical operational metrics for troubleshooting and analysis. Ambari predefines alerts which integrate with existing enterprise monitoring tools that monitor cluster components and hosts as per specified check intervals. Through the browser interface users can browse alerts for their clusters, search and filter alerts. They can also view and modify alert properties alert instances associated with that definition.

4.Metrics visualization and dashboarding

In this Apache Ambari tutorial you can know that it provides scalable low latency storage system for Hadoop component metrics. To pick the metrics of Hadoop which truly matter requires considerable expertise and understanding on how the components work with each other and with themselves. Grafana is a leading graph and dashboard builder which simplifies the metrics reviewing process. This is included with Ambari metrics along with HDP.

5.Extensibility and customization

Ambari lets a developer to work on Hadoop gracefully in one's enterprise setup. Ambari leverages the large innovative community which improve upon the tool and it also eliminates vendor lock in. REST APIs along with Ambari Stacks and Views allows extensive flexibility for customization of HDP implementation.

	Ambari Stacks wrap lifecycle control layer to rationalize operations over a broad set of services. This includes a consistent approach which the Ambari technology uses to manage different types of services like install, start, configure, status, stop. When provisioning, cluster install experience is rationalized across a set of services by Stacks technology. A natural extension point for operators is provided by the Stacks to plug in newly created services that can perform alongside Hadoop. Third parties can plug in their views through Ambari views. A view is an application that is deployed into Ambari container where it offers UI capabilities to be plugged in to give out custom visualization, management and monitoring			
8	**Explain how Zookeeper works. How is Apache Ambari different from Zookeeper?** **Solution:** ZooKeeper, while being a coordination service for distributed systems, is a distributed application on its own. ZooKeeper follows a simple client-server model where clients are nodes (i.e., machines) that make use of the service, and servers are nodes that provide the service. A collection of ZooKeeper servers forms a ZooKeeper ensemble. Once a ZooKeeper ensemble starts after the leader election process, it will wait for the clients to connect. At any given time, one ZooKeeper client is connected to one ZooKeeper server. Each ZooKeeper server can handle a large number of client connections at the same time. Each client periodically sends pings to the ZooKeeper server it is connected to let it know that it is alive and connected. The ZooKeeper server in question responds with an acknowledgment of the ping, indicating the server is alive as well. When the client doesn't receive an acknowledgment from the server within the specified time, the client connects to another server in the ensemble, and the client session is transparently transferred over to the new ZooKeeper server.  ZooKeeper has a file system-like data model composed of znodes. Think of znodes (ZooKeeper data nodes) as files in a traditional UNIX-like system, except that they can have child nodes. Another way to look at them is as directories that can have data associated with themselves. Each of these directories is called a znode. The znode hierarchy is stored in memory within each of the ZooKeeper servers. This allows for scalable and quick responses to reads from the clients. Each ZooKeeper server also maintains a transaction log on the disk, which logs all write requests. ZooKeeper server must sync transactions to disk before it returns a successful response. The default maximum size of data that can be stored in a znode is 1 MB. Zookeeper should only be used as a storage mechanism for the small amount of data required for providing reliability, availability, and coordination to your distributed application. **Features:** a. Fast. ZooKeeper is especially fast with workloads where reads to the data are more common than writes. The ideal read/write ratio is about 10:1. b. Reliable. ZooKeeper is replicated over a set of hosts (called an ensemble) and the servers are aware of each other. As long as a critical mass of servers is available, the ZooKeeper service will also be available. There is no single point of failure.	[10]	CO2	L2

c. Simple. ZooKeeper maintain a standard hierarchical name space, similar to files and directories.
d. Ordered. The service maintains a record of all transactions, which can be used for higher-level abstractions, like synchronization primitives.

How is Ambari different from ZooKeeper

This description may confuse you as Zookeeper performs the similar kind of tasks. But, there is a huge difference between the tasks performed by these two technologies if looked closely.

Basis of Difference	**Apache Ambari**	**Apache ZooKeeper**
Basic Task	Monitoring, provisioning and managing Hadoop cluster	Maintaining configuration information, naming and synchronizing the cluster.
Nature	Web interface	Open-source server
Status maintenance	Status maintained through APIs	Status maintained through znodes