Hadoop - Knowledge Purpose:

Question 1:

A) *Commissioning and Decommissioning in terms of Hadoop data nodes:*

- Commissioning of nodes stands for adding new nodes in the current cluster which operates your Hadoop framework.
- In contrast, Decommissioning of nodes stands for removing nodes (faulty) permanently or for maintenance from your cluster.

Steps involved in Decommissioning Nodes:

a. Update the network addresses in the 'exclude' files **dfs.exclude mapred.exclude**
b. Update the NameNode: **hadoop dfsadmin -refreshNodes**
c. Update the JobTracker: **hadoop mradmin -refreshNodes**
d. Cross Check the Web UI to ensure the successful removal **"Decommission in Progress" "Decommissioned"**
e. Remove the Nodes from include file and run: **hadoop dfsadmin -refreshNodes**
f. Remove the Nodes from 'slaves' file

Steps involved in Commissioning Nodes:

a. Update the network addresses in the 'include' files **dfs.include mapred.include**
b. Update the NameNode: **hadoop dfsadmin -refreshNodes**
c. Update the JobTracker: **hadoop mradmin -refreshNodes**
d. Update the 'slaves' file
e. Start the DataNode and TaskTracker **hadoop-deamon.sh start tasktracker hadoop-deamon.sh start datanode**
f. Cross Check the Web UI to ensure the successful addition
g. Run Balancer to move the HDFS blocks to DataNodes

Practical Example where Commissioning and Decommissioning are used:

- As an administrator, If the organization is growing, we need to add more nodes to our cluster to meet the SLAs (Service Level Agreements) or, sometimes due to maintenance activity, you may need to take down a certain node.

B) *Difference between Secondary NameNode and Standby NameNode:*

Secondary NameNode:

In Hadoop 1.x and 2.x, the secondary NameNode means the same. It does CPU intensive tasks for NameNode. In more details, it combines the Edit log and fs_image (meta data) and returns the consolidated file to NameNode. NameNode then loads that file into RAM. But, the secondary NameNode doesn't provide failover capabilities (i.e.,

doesn't act as Primary NameNode). So, in case of NameNode failure, data is retained from Secondary Namenode.

Standby NameNode:
In Hadoop 2.x, the Standby NameNode came into picture. The Standby NameNode is the node that removes the problem of SPOF (Single Point Of Failure) that was there in Hadoop 1.x. The Standby NameNode provides automatic failover in case Active Namenode fails (i.e., can act as NameNode)

Secondary NameNode Contains:
It typically stores Metadata from the NameNode. The meta data consists of two files:

- **FsImage file:** It is a snapshot of the HDFS file system metadata at a certain point of time. That is, typically stores all filenames, locations and block structures
- **EditLog file:** It is a transaction log which contains records for every change (such as modifications and transactions related updates) that occurs to file system metadata.

C) *Steps to be followed when Primary NameNode Crashes:*

- Use the file system metadata replica (FsImage) to start a new NameNode.
- Then, configure the DataNodes and clients so that they can acknowledge this new NameNode, that is started.
- Now the new NameNode will start serving the client after it has completed loading the last checkpoint FsImage (for metadata information) and received enough block reports from the DataNodes.

Question 2:

A) *Primary difference between Hadoop 2.x and 3.x architectures*

Hadoop 2.x:

- Minimum Java Version requirement: Java 7
- Fault tolerance is done by replication only
- YARN timeline service has Scalability Issues
- HDFS has 200% overhead in storage space
- DataNode resource is not dedicated for MapReduce we can use it for other applications
- Introduced with master daemon termed as Standby NameNode

Hadoop 3.x:

- Minimum Java Version requirement: Java 8
- Fault Tolerance is done by erasure coding
- YARN timeline service is improved. Scalability and reliability of service is improved
- Storage overhead is only 50%
- Also DataNode resources can be used for other applications too

- Introduced with Multiple Standby NameNodes to boost availability

B) Hadoop vs Spark:

Hadoop:

- Performance is slow
- Written in Java
- Generally for OLAP (Online Analytical Processing)
- Less level of Abstraction
- Uses Batch processing with huge volume of data
- High Latency Computing
- Process data in batch mode
- Highly secure
- Complex and Lengthier codes
- Cannot handle Graph processing
- Doesn't support Caching of Data

Spark:

- 100x Performance than Hadoop
- Written in SCALA
- Generally for OLTP (Online Transaction Processing)
- High level of Abstraction
- Process Real-time data
- Low Latency Computing
- Can Process data interactively
- Less Secure compared to Hadoop
- Compact and Simple Compared to Hadoop
- Can handle Graph Processing
- Supports Caching of Data

C) Hadoop vs RDBMS:

RDBMS:

- Static Schema: Required on Write
- Structured Data type
- Stored in logical form with inner-related tables and columns
- No Replication
- Works with data with size in terms of GB
- Interactive and Batch Processing
- Vertical Scaling can be done (adding hardware)
- Uses Licensed Software tools
- Read and Write Multiple times

Hadoop:

- Dynamic Schema: Required on Read
- Multi and Unstructured Data Type
- Stored in Compression Format

- Replication is allowed for high scalability and availability under all circumstance
- Works with data with size in terms of PB
- Batch Processing
- Horizontal Scaling (adding machines into clusters)
- Free and Open Source
- Write once, Read Multiple times

D) Different Data Types in Hadoop:

- **IntWritable:** Hadoop variant of Integer
- **VIntWritable:** Variable length Integer types
- **FloatWritable:** Hadoop variant of Float
- **LongWritable:** Hadoop variant of Long
- **VLongWritable:** Variable length Long types
- **ShortWritable:** Hadoop variant of Short
- **DoubleWritable:** Hadoop variant of Double
- **Text:** Hadoop variant of String
- **ByteWritable:** Hadoop variant of Byte
- **NullWritable:** Hadoop variant of null

Writable and WritableComparable Class:

Writable Class:

- Meant for writing the data to the local disk.
- These Writable data types are passed as parameters for the mapper and reducer.

WritableComparable Class:

- It is a combination of Writable and Comparable classes.
- Comparable is used for comparing when the reducer sorts the keys, and Writable can write the result to the local disk.

Question 3:

A) Benefits of Spark over MapReduce:

- Spark is easy to program and doesn't require that much of hand coding whereas MapReduce is not that easy in terms of programming and requires lots of hand coding
- Spark has interactive mode whereas in MapReduce there is no built-in interactive mode, MapReduce is developed for batch processing.
- Spark executes batch processing jobs about 10 to 100 times faster than Hadoop MapReduce.
- Spark uses an abstraction called RDD which makes Spark feature rich, whereas map reduce doesn't have any abstraction
- Spark uses lower latency by caching partial/ complete results across distributed nodes whereas MapReduce is completely disk-based.

B) *RDD in Spark:*

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. At the core, an RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster that can be operated in parallel with a low-level API that offers transformations and actions.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. It supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is shareable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

How to create RDD in Spark:

- Parallelizing already existing collection in driver program.
- Referencing a dataset in an external storage system (e.g. HDFS, Hbase, shared file system).
- Creating RDD from already existing RDDs.

C) *Lazy Evaluation in Spark:*

Lazy Evaluation means that the execution will not start until an action is triggered. In Spark, the picture of lazy evaluation comes when Spark transformations occur. Transformations are lazy in nature meaning when we call some operation in RDD, it does not execute immediately. Spark maintains the record of which operation is being called (through DAG). We can think of Spark RDD as the data that we built up through transformation. Since transformations are lazy in nature, we can execute operations any time by calling an action on data. Hence, in lazy evaluation data is not loaded until it is necessary.

Advantages:

- Increases Manageability
- Saves Computation and Increase Speed
- Reduces Complexities
- Provides Optimization

D) *Key Features of Apache Spark:*

- Speed: Runs an application up to 100 times faster in memory, and 10 times faster when running on disk.
- Supports Multiple Languages: Provides built-in API in Java, Scala or Python
- Advanced Analytics: Not only supports MapReduce, also supports SQL queries, streaming data, ML, and Graph Algorithms

E) Various Levels of Persistence in Apache Spark:

- **MEMORY_ONLY**: In this level, RDD object is stored as a de-serialized Java Object in Java Virtual Machine (JVM). If an RDD doesn't fit in the memory, it will be recomputed.
- **MEMORY_AND_DISK**: In this level, RDD object is stored as a de-serialized Java object in JVM. If an RDD doesn't fit in the memory, it will be stored on the Disk.
- **MEMORY_ONLY_SER**: In this level, RDD object is stored as a serialized Java object in JVM. It is more efficient than de-serialized objects.
- **MEMORY_AND_DISK_SER**: In this level, RDD object is stored as a serialized Java object in JVM. If an RDD doesn't fit in the memory, it will be stored on the Disk.
- **DISK_ONLY**: In this level, RDD objects are stored only on Disk.