1) **If 8TB is the available disk space per node (10 disks with 1 TB, 2 disk for operating system etc. were excluded.). Assuming initial data size is 600 TB. How will you estimate the number of data nodes (n)?**

Estimating the hardware requirement is always challenging in the Hadoop environment because we never know when data storage demand can increase for a business. We must understand following factors in detail to come to a conclusion for the current scenario of adding right numbers to the cluster:

- The actual size of data to store – 600 TB
- At what pace the data will increase in the future (per day/week/month/quarter/year) – Data trending analysis or business requirement justification (prediction)
- We are in Hadoop world, so replication factor plays an important role – default 3x replicas
- Hardware machine overhead (OS, logs etc.) – 2 disks were considered
- Intermediate mapper and reducer data output on hard disk - 1x
- Space utilization between 60 % to 70 % - Finally, as a perfect designer we never want our hard drives to be full with their storage capacity.
- Compression ratio

Let's do some calculation to find the number of data nodes required to store 600 TB of data:

Rough calculation:

Data Size – 600 TB
Replication factor – 3
Intermediate data – 1
Total Storage requirement – (3+1) * 600 = 2400 TB
Available disk size for storage – 8 TB
Total number of required data nodes (approx.): 2400/8 = 300 machines
Actual Calculation: Rough Calculation + Disk space utilization + Compression ratio

Disk space utilization – 65 % (differ business to business)
Compression ratio – 2.3
Total Storage requirement – 2400/2.3 = 1043.5 TB
Available disk size for storage – 8*0.65 = 5.2 TB
Total number of required data nodes (approx.): 1043.5/5.2 = 201 machines
Actual usable cluster size (100 %): (201*8*2.3)/4 = 925 TB
Case: Business has predicted 20 % data increase in a quarter and we need to predict the new machines to be added in a year

Data increase – 20 % over a quarter
Additional data:
1st quarter: 1043.5 * 0.2 = 208.7 TB
2nd quarter: 1043.5 * 1.2 * 0.2 = 250.44 TB
3rd quarter: 1043.5 * (1.2)^2 * 0.2 = 300.5 TB
4th quarter: 1043.5 * (1.2)^3 * 0.2 = 360.6 TB
Additional data nodes requirement (approx.):
1st quarter: 208.7/5.2 = 41 machines

2nd quarter: 250.44/5.2 = 49 machines
3rd quarter: 300.5/5.2 = 58 machines
4th quarter: 360.6/5.2 = 70 machines
With these numbers you can predict next year additional machine requirements for the cluster (last quarter + 24), (last quarter + 28) and so on.

2.) **Imagine that you are uploading a file of 500MB into HDFS.100MB of data is successfully uploaded into HDFS and another client wants to read the uploaded data while the upload is still in progress. What will happen in such a scenario, will the 100 MB of data that is uploaded will be displayed?**

Although the default blocks size is 64 MB in Hadoop 1x and 128 MB in Hadoop 2x whereas in such a scenario let us consider block size to be 100 MB which means that we are going to have 5 blocks replicated 3 times (default replication factor). Let's consider an example of how does a block is written to HDFS:

We have 5 blocks (A/B/C/D/E) for a file, a client, a namenode and a datanode. So, first the client will take Block A and will approach namenode for datanode location to store this block and the replicated copies. Once the client is aware about the datanode information, it will directly reach out to datanode and start copying Block A which will be simultaneously replicated to other 2 datanodes. Once the block is copied and replicated to the datanodes, the client will get the confirmation about the Block A storage and then, it will initiate the same process for the next block "Block B".

So, during this process if the 1st block of 100 MB is written to HDFS and the next block has been started by the client to store then the 1st block will be visible to readers. Only the current block being written will not be visible by the readers.

3) When does a NameNode enter the safe mode?

Namenode is responsible for managing the meta storage of the cluster and if something is missing from the cluster then Namenode will be held. This makes Namenode check all the necessary information during the safe mode before making the cluster writable to the users. There are couple of reasons for Namenode to enter the safe mode during startup such as;

i) Namenode loads the filesystem state from fsimage and edits log files, it then waits for datanodes to report their blocks, so it does not start replicating the blocks which already exist in the cluster.

ii) Heartbeats from all the datanodes and also if any corrupt blocks exist in the cluster. Once Namenode verifies all this information, it will leave the safe mode and make the cluster accessible. Sometimes, we need to manually enter/leave the safe mode for Namenode which can be done using the command line "hdfs dfsadmin -safemode enter/leave".

4) What are the steps followed by the application while running a YARN job when calling a SubmitApplication method?

All jobs that are submitted by a client go to the resource manager. The resource manager is provided with a scheduler, and it is the responsibility of the resource manager to determine the resources required to run that particular job.

Once the resources and the number of resources are determined, the resource manager will then launch the application masters specific to the applications that are to be run.

The application master associated with a specific application, also known as the application master daemon remains available until the job gets completed.

The duty of the application master is to negotiate resources from the resource manager. This means that the application manager will ask for the number of resources as specified by the resource manager.

The application manager will then launch the container in a different node manager once the data becomes available.

Node managers monitor the containers, and a node manager is responsible for all containers available in that particular node. The container is responsible for giving periodic updates to the application manager regarding the job that it is executing.

Once the job gets completed, the container and the resources get freed up, following which the application manager proceeds to update the resource manager that the job is completed. The client then receives the corresponding update from the resource manager.

5) Suppose you want to get an HDFS file into a local directory; how would you go about it?

There are two commands that can be used to get HDFS files into the local system:

```
hadoop fs -get
hadoop fs -copyToLocal
```

6) What command will you use to copy data from one node in Hadoop to another?

```
hdfs dfs -distcp hdfs://source_namenode/apache_hadoop hdfs://dest_namenodeB/Hadoop
```

7) How can you kill an application running on YARN?

Use:

```
yarn application -list
```

To list all the applications that are running on YARN.

To kill the application that you want to kill, identify its application ID and use the following command to kill it:

```
yarn application -kill appid
```

8) In MapReduce tasks, each reduce task writes its output to a file named part-r-nnnnn. Here nnnnn is the partition ID associated with the reduce task. Is it possible to ultimately merge these files? Explain your answer.

The files do not get automatically merged by Hadoop. The number of files generated is equal to the number of reduced tasks that take place. If you need that as input for the next job, there is no need to worry about having separate files. Simply specify the entire directory as input for the next job. If the data from the files must be pulled out of the cluster, they can be merged while transferring the data.

The following command may be used to merge the files while pulling the data off the cluster:

```
Hadoop fs -cat //part-r-* > /directory of destination path
```
The complete merging of the output files from the reduce tasks can be delegated using the following command as well:

```
Hadoop fs -getmerge / /
```

9) There is a YARN cluster in which the total amount of memory available is 40GB. There are two application queues, ApplicationA and ApplicationB. The queue of ApplicationA has 20 GB allocated, while that of ApplicationB has 8GB allocated. Each map task requires an allocation of 32GB. How will the fair scheduler assign the available memory resources under the DRF (Dominant Resource Finder) Scheduler?

The allocation of resources within a particular queue is controlled separately. Within one queue:

The FairScheduler can apply either the FIFO policy, the FairPolicy or the DominantResourceFairnessPolicy. The CapacityScheduler may use either the FIFOPolicy or the FairPolicy.

The default scheduling policy of the FairScheduler is the Fair policy, where memory is used as a resource.The DRF policy uses both memory and CPU as resources and allocates them accordingly. DRF is quite similar to fair-scheduling. However, the difference is that DRF primarily applies to the allocation of resources among queues. This is already heavily handled by queue weights. Hence, the most crucial job of the DRF is to manage multiple resources rather than equal resource allocation.

In this case, initially, both Application A And Applicaton B will have some resources allocated to various jobs present in their corresponding queues. In such a way, only 12GB (40GB - (20 GB + 8 GB)) will remain in the cluster. Each of the queues will request to run a map task of size 32GB. The total memory available is 40 GB.The rest of the required resources can be taken from the CPU. In such a case, ApplicationA currently holds 20GB. Another 12GB is required for the map task to get executed. Here, the fair scheduler will grant the container requesting 12GB of memory to ApplicationA. The memory allocated to ApplicationB is 8GB, and it will require another 24 GB to run a map task. Memory is not available for application, and hence the DRF will try to use 8 GB from memory, and the remaining 20GB will be used from the CPU.

10) How does a NameNode know that one of the DataNodes in a cluster is not functioning?

Hadoop clusters follow a master-slave architecture, in which the NameNode acts as the master and the Data Nodes act as the slaves. The data nodes contain the actual data that has to be stored or processed in a cluster. The data nodes are responsible for sending heartbeat messages to the NameNode every 3 seconds to confirm that they are active or alive. Suppose the NameNode fails to receive a heartbeat message from a particular node for more than ten minutes. In that case, the NameNode considers that particular data node to be no longer active or dead. The name node then initiates replication of the data on the dead data node blocks to some of the other data nodes that are active on the cluster. Data nodes are able to talk to each other to rebalance the data and their replicas within a cluster. They can copy data around and transfer it to keep the replication valid in the cluster. The data nodes store metadata and information about which files can be mapped to which particular block location. Data nodes also maintain a checksum for each block. When data gets written to HDFS, the checksum value is written simultaneously to the data node. When the data gets read, by default, the same checksum value is used for verification.

Data nodes are responsible for updating the name node with the block information at regular intervals of time and before verifying the checksum's value. If the checksum value is not correct for a specific block, then that particular block can be considered to have a disk-level corruption. Since there is an issue in the reporting of the block information to the name node, the name node is able to know that there is a disk-level corruption on the data and necessary steps have to be taken to copy the data to alternate locations on other active data nodes to maintain the replication factor.