1. **For NameNode, why it’s not necessary to store block locations persistently?**

An important feature of the design is that data is never moved through the NameNode. All data transfer occurs directly between clients and DataNodes; communications with the NameNode only involves transfer of metadata.

When a client requests to read or write a file, the NameNode provides the block locations, and the client directly contacts the corresponding DataNodes to access the data.

Therefore, the NameNode does not need to store block locations persistently and can instead rely on the DataNodes to provide the information when needed.

1. **Why is it important to make the NameNode resilient to failures?**

NameNode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. The client communicates with HDFS with the help of NameNode.

If the NameNode fails, the entire HDFS cluster becomes unavailable, and clients cannot access their data, resulting in significant downtime, data loss, and impact on business operations that rely on the HDFS cluster.

By making the NameNode resilient to failures, the availability and reliability of the HDFS cluster are improved, and the impact of failures on the business operations that rely on it is minimized.

1. **What details are there in the FsImage file?**

FsImage file is a complete persistent checkpoint of the filesystem metadata (point in time snapshot).

1. **What is the purpose of the Secondary NameNode?**

The Secondary NameNode in Hadoop Distributed File System (HDFS) is a helper node that performs two important functions for the primary NameNode:

* + Periodically creates a new checkpoint: The Secondary NameNode takes a snapshot of the file system metadata (fsImage) from the primary NameNode and combines it with the edit log to create a new checkpoint. This helps to reduce the time it takes to recover from a NameNode failure because the primary NameNode can use the new checkpoint to restart quickly instead of having to replay the entire edit log.
  + Helps to manage the size of the edit log: The edit log records all the changes made to the file system metadata since the last checkpoint. Over time, the edit log can become very large, making a recovery slower and more difficult. The Secondary NameNode periodically merges the edit log with the previous checkpoint to create a new one, which helps reduce the edit log's size.

Overall, the Secondary NameNode plays an important role in maintaining the health and stability of the HDFS cluster by helping to reduce the recovery time in case of failures and managing the size of the edit log.

1. **Does the NameNode stay in the safe mode until all under-replicated files are fully replicated? Why or why not?**

Yes, It does.

In HDFS, the NameNode enters the safe mode when the cluster starts up or when the cluster experiences a problem, such as a failure of one or more DataNodes. The purpose of the safe mode is to prevent data loss by ensuring that all data is properly replicated and available before allowing any modifications to the files in the cluster.

When the NameNode enters safe mode, it checks the status of all the blocks in the cluster and marks any under-replicated blocks as being in a special state called "under-replicated mode." In this mode, the NameNode tracks the progress of the replication process for each block and waits until all under-replicated blocks have been fully replicated before exiting safe mode.

1. **What are the core changes in Hadoop 2.x compared to Hadoop 1.x? In other words, state the major differences between Hadoop 1 and Hadoop 2.**

Horizontal scaling capability is added in Hadoop 2 with HDFS Federation in which you can have multiple NameNodes taking care of multiple namespaces in a single Hadoop cluster. NameNodes do share the Block Pool which is the storage of all the DataNodes.

* + High Availability – Taking care of NameNode SPOF problem.
  + YARN - Taking care of JobTracker SPOF and added support for non-mapreduce type of processing (multitenancy) making MapReduce as a user library, or one of the applications residing in Hadoop.
  + HDFS Federation - Added support for multiple namespaces with multiple NameNodes.
  + High Cluster Utilization - Use of variable-sized Containers instead of fixed-size Slots mechanism
  + Improved Scalability - Hadoop 2.x supports more than 10,000 nodes per cluster.
  + MRv2 (simply MRv1 rewritten to run on top of YARN) – no need to rewrite existing MapReduce jobs.
  + Beyond Java

1. **What is the difference between MR1 in Hadoop 1.0 and MR2 in Hadoop2.0?**

MRv2 simply MRv1 rewritten to run on top of YARN.

1. **What is HDFS Federation? What advantage does it provide?**

HDFS Federation is a feature of HDFS that enables multiple NameNodes to manage separate portions of the file system namespace, each with its own set of DataNodes. This provides several advantages over a traditional HDFS cluster with a single NameNode.

HDFS Federation provides a way to scale HDFS clusters beyond the limits of a single NameNode and provides greater reliability, isolation, and flexibility for managing large-scale data processing workloads.

1. **What is NameNode High Availability and how is it achieved in Hadoop 2?**

The High Availability (HA) feature in Hadoop 2 addresses the NameNode SPOF problem by providing the option of running two redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby.

This allows a fast failover to a new NameNode in the case that a machine crashes, or a graceful administrator-initiated failover for the purpose of planned maintenance.

In the event that the active NameNode fails, the standby NameNode will take over as the active NameNode.

This failover can be configured to be automatic, negating the need for human intervention. The fact that a NameNode failover occurred is transparent to Hadoop clients.

In a typical HA cluster, two separate machines are configured as NameNodes. At any point in time, exactly one of the NameNodes is in an Active state, and the other is in a Standby state.

The Active NameNode is responsible for all client operations in the cluster, while the Standby is simply acting as a slave, maintaining enough state to provide a fast failover if necessary.

In order for the Standby node to keep its state synchronized with the Active node, the current implementation requires that the two nodes both have access to a shared directory on a shared storage device.

When any namespace modification is performed by the Active node, it durably logs a record of the modification to an edits log file stored in the shared directory.

The Standby node is constantly watching this directory for edits, and as it sees the edits, it applies them to its own namespace.

In the event of a failover, the Standby will ensure that it has read all of the edits from the shared storage before promoting itself to the Active state.

This ensures that the namespace state is fully synchronized before a failover occurs.

In order to provide a fast failover, it is also necessary that the Standby node has up-to-date information regarding the location of blocks in the cluster.

In order to achieve this, the DataNodes are configured with the location of both NameNodes and send block location information and heartbeats to both.

1. **What is the role of Application Master in YARN application execution?**

The AM is responsible for coordinating the allocation of resources for a specific application and managing its execution.

When an application is submitted to YARN, the ResourceManager (RM) assigns an Application Master to manage its execution. The AM is launched on a NodeManager (NM) node, and its primary role is to communicate with the ResourceManager to request resources for the application.

Once the AM has obtained the necessary resources, it is responsible for coordinating the execution of the application tasks. This involves launching container processes on the allocated resources, monitoring their execution, and reporting back to the ResourceManager on their status.

The AM is also responsible for handling application failures and retries. If a task fails, the AM can request that the ResourceManager allocate a new container and re-launch the task. The AM is also responsible for monitoring the progress of the application and reporting back to the client.

Overall, the AM plays a crucial role in the execution of YARN applications, managing the allocation of resources and tasks, and ensuring that the application runs efficiently and reliably.