Hadoop:

1. What are the differences between regular Filesystem and HDFS?

Regular file system is stores data as single copy. Hence it is not fault tolerant.

HDFS divides data into multiple blocks and stores data into different data nodes.

Also data is replicated across multiple data nodes making it highly fault tolerant and reliable.

Block size is much higher in HDFS in comparison to regular file system

Regular file system follows tree format to store the data. HDFS on the other hand is distributed file system and uses master slave architecture to store the data.

Data retrieval in regular file system is slower as compared to HDFS.

HDFS is tuned to support large files. Regular file system is not tuned for that.

Applications that run on HDFS have large data sets.

HDFS is designed to be portable across heterogenous hardware and software platforms unlike regular file system which is OS specific.

1. Why is HDFS Fault tolerant.

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. HDFS is fault-tolerant because it replicates data on different DataNodes. By default, a block of data is replicated on three DataNodes.

The data blocks are stored in different DataNodes. If one node crashes, the data can still be retrieved from other DataNodes.

1. Architecture of HDFS:

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system’s clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode

Input Split is logical split of data, basically used during data processing in MapReduce program or other processing techniques.

Input Split size is user defined value and Hadoop Developer can choose split size based on the size of data(How much data you are processing).Input Split is basically used to control number of Mapper in MapReduce program.

If you have not defined input split size in MapReduce program then default HDFS block split will be considered as input split during the data processing.

So for a input file of 350 MB considering a default block size of 128 MB it will be split into 3 blocks (block size 128 MB). If we have not specified the input split size

then there will be 3 input splits meaning 3 mappers.

By changing **mapred.min.split.size in** **mapred-site.xml** we can change the input split size

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

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

1. Two many small files create performance bottleneck in HDFS.

A file which is less than HDFS block size(64MB/128MB) is termed as small file.

Namenode stores all files metadata in memory, so if you are storing lots of small files,namenode has to maintain its metadata, for a file metadata, it occupies 150 bytes so in the case of million files it would cost around 3GB of memory.

Though it keeps the persistant copy of metadata in a disk, it needs to store the metadata in memory for fast retreival.

So, because of small files, it would hamper the MapReduce computation also.

If we create two many small files any application then accessing these files will take considerably longer time to scan through the files and process data.

There are many ways we can address too many small files problem.

For example if we are processing data using Spark we can use coalesce to reduce the no of partitions . Alternatively we can set the spark.sql.shuffle.partitions property to a sensible value to reduce the no of part files created by spark.

Another way to tackle the problem is to use Sequence Files where you use the filename as the key and the file contents as the value.

You can create MapReduce program convert lots of small files to into a single SequenceFile.SequenceFiles are splittable, so MapReduce can break them into chunks and operate on each chunk independently.They support block compression which is the best option.

Hive

1. What is the difference between Internal and External tables?

Internal tables are also known as Managed tables that are owned and managed by Hive. By default, Hive creates a table as an Internal table and owned the table structure and the files.

In other words, Hive completely manages the lifecycle of the table (metadata & data) similar to tables in RDBMS.

For Internal tables, Hive by default stores the files at the data warehouse location which is located at /user/hive/warehouse

When you drop an internal table, it drops the data and also drops the metadata of the table.

Data in External tables are not owned or managed by Hive. To create an External table you need to use EXTERNAL clause.

Hive default stores external table files also at Hive managed data warehouse location but recommends to use external location using LOCATION clause.

Dropping an external table just drops the metadata but not the actual data. The actual data is still accessible outside of Hive.

1. We will not be able to access the data directly. We will first need to do a MSCK REPAIR on the table before we can start accessing/reading the data.
2. If we have created the table as a managed table then yes when we drop the partition the data will be deleted from HDFS.
3. SCD in Hive:

For enabling SCD in Hive we need to make the target table ACID enabled that means we need to create manged table in ORC format with TBLPROPERTIES ('transactional'='true')

We can get the incremental data in an external table in the staging area.

Post that we can use the merge into target table by joining the staging and target tables we can achieve SCD type1

merge into

target\_table target

using

stage\_table stage

on

stage.id = target.id

when matched then

update set target.field1= stage.field1 ,target.field2=stage.field2

when not matched then

insert values (sstage.id,tage.ifield1,stage.field2);

For SCD Type2:

merge into target\_table target

using (

select

stage.id as join\_key,

stage.\* from stage\_table stage

union all

select

null, stage\_table.\*

from

stage\_table join target\_table

on stage\_table.id = target\_table.id

where

( stage\_table.field1 <> target\_table.filed1

or stage\_table.field2 <> target\_table.filed2)

and target\_table.valid\_to is null

) upd

on upd.join\_key = target.id

when matched

and upd.field1 <> target.field1 or upd.field2 <> target.field2

then update set valid\_to = current\_date()

when not matched

then insert

values (upd.id, upd.field1 , upd.field2, current\_date(), null);

IF SCD is not preferred way considering performance we can provide other alternatives

1. We can look at creating a partition by load\_date if we are loading high volume of data every single day and then we can restrict our query to use the partitioned column load\_date to query it faster

SQL

**select** \* FROM ( **select**  **Salary** ,rank() over (order by **Salary** DESC) rnk from Employee ) WHERE rnk = N;

1. DELETE A

FROM table A

INNER JOIN

(

SELECT \*,

rank() OVER(PARTITION BY pk fields,

ORDER BY id) rnk

FROM table

) B ON A.ID = B.ID

WHERE rnk > 1;

1. INNER JOIN: Returns records that have matching values in both tables

LEFT OUTER JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT OUTER JOIN: Returns all records from the right table, and the matched records from the left table

FULL OUTER JOIN: Returns all records when there is a match in either left or right table

1. Database normalization is the process of structuring a database in accordance with a series normal forms in order to reduce data redundancy and improve data integrity.

The goal of normalization is to remove any duplicates that might appear within the data set.Redundancies can adversely affect analysis of data since they are values which aren’t exactly needed.

Expunging them from the database helps to clean up the data, making it easier to analyze.Another step is resolving any conflicting data. Sometimes, datasets will have information that conflicts with each other,

so data normalization is meant to address this conflicting issue and solve it before continuing. A third step is formatting the data. This takes data and converts it into a format that allows further processing and analysis to be done. Finally,

data normalization consolidates data, combining it into a much more organized structure.

1. Union and Union all differences:

The difference between UNION and UNION ALL is that UNION will omit duplicate records whereas UNION ALL will include duplicate records.

Union all will result in duplicate records. Union all is faster as it does not do a distinct

1. RANGE partitioning. This type of partitioning assigns rows to partitions based on column values falling within a given range. ( PARTITION p0 VALUES LESS THAN (100),

PARTITION p1 VALUES LESS THAN (200),

PARTITION p2 VALUES LESS THAN (300),

PARTITION p3 VALUES LESS THAN (400))

LIST partitioning. Similar to partitioning by RANGE, except that the partition is selected based on columns matching one of a set of discrete values.

( PARTITION p0 VALUES (1,4,6,8),

PARTITION p1 VALUES (7,11,13))

Hash Partitioning: Hash partitioning can be used if the columns of a table are not suitable for Range or List partitioning.Hash partitioning divides partitions using a hashing algorithm that partitioning key is applied that is identified.

# Shell

1. Write a Shell Script that adds two numbers if provided as the command Line Argument and if the two numbers are not entered it outputs an Error Message along with a one-Line of how-to use description.

#!/bin/bash

if [ "$#" -ne 2 ]; then

echo "Incorrect way of running script. Example invocation sh addnumbers.sh 123 321 "

exit 1

fi

num1=$1

num2=$2

sum=$(($num1 + $num2))

echo $sum

exec &> addnumbers.log.$(date '+%Y%m%d')

1. Write a Shell script to clean up a CSV file for junk characters.

#!/bin/bash

sed -i 's/[\d128-\d255]//g' input\_\_file\_location/input\_file.csv

1. Write a shell script to print a given Number say 10572, in reverse order such that the input is provided using command Line Argument only. If the input data is not provided as Command Line Argument, it should throw and error and should suggest, how to use the script.

#!/bin/bash

if [ "$#" -ne 1 ]; then

echo "Incorrect way of running script. Example invocation sh reversenum.sh 4567 "

exit 1

fi

num=$1

echo $num | rev