Q. Explain in Brief.

(A) Sequence File Format

1) Sequence file format is one of the hadoop specific file format and stores the serialized data hence the transferring of the file becomes very easy if stored in the sequence file format.

2) Hadoop does not work very well with a lot of small files hence in order to solve this the sequence files are used as the container to store the small files and convert them into one large file.

3) Sequence files are flat files containing key, value pairs.

4) Every Sequence file has got a header which describes the about the file. The header contains information such as key/value class names, version, file format, metadata about the file and sync marker to denote the end of the header.

5) Sequence file can be compressed so as to take up less space. This compression codec is also mentioned in the header.

6) Sequence file format can be compressed in three ways

a) Uncompressed

b) Record compressed

c) Block compressed

7) Different compression formats are avro, lz4, gz etc.

(B) Nline Input Format

It which splits N lines of input as one split.

job.setInputFormatClass(NLineInputFormat.class);

NLineInputFormat.addInputPath(job, inputPath);

job.getConfiguration().setInt("mapreduce.input.lineinputformat.linespermap", 1);

Above lines can be used to set the how many lines are considered for one split . Here it is set to one. We can modify to any number one wants.

Hence fixed number of lines are sent to each mapper.

NLineInputFormat is used in applications that take a small amount of input data and run an extensive (that is, CPU-intensive) computation for it, then emit their output.

Another example of nline input is seeding of the data from different sources.

We can create a “seed” input file that lists the data sources, one per line. Then each mapper is allocated a single data source, and it loads the data from that source into HDFS.

Default value is always set to one.

(C) and (D) DB Input and DB output Format

Database input format allows hadoop to input to output data to any relational database systems. Allowing relational data to be more easily incorporated into your data processing pipeline.

DB Input Format uses the JDBC to connect to the sources as it is widely used.

DBInputFormat can work with MySQL, PostgreSQL, and several other database systems.

To start using DBInputFormat to connect to your database, you’ll need to download the appropriate database.

These are the changes that are to be made in driver to use db input format

DBConfiguration.configureDB(<Configuration>,<Driver>,<Connection String>,<UserName>,<Password>);

job.setInputFormatClass(DBInputFormat.class);

DBOutputFormat.setOutput(<Job>,<Output Table>,<List of table columns>);

Changes in mapper code

public class DBInputFormatMapextends Mapper<LongWritable, DBInputWritable, Text, NullWritable>

protected void map(

LongWritable id,

DBInputWritable value, Context context)

Implementation of the db input format.

public void

readFields(ResultSet rs) throws SQLException

{

id = rs.getInt(1);

name = rs.getString(2);

location = rs.getString(3);

}

Similar modifications have to be done in db output format

Driver code

DBConfiguration.configureDB(<Configuration>,<Driver>,<Connection String>,<UserName>,<Password>);

job.setInputFormatClass(DBInputFormat.class);

DBOutputFormat.setOutput(<Job>,<Output Table>,<List of table columns>);

Mapper code

public class DBOutputFormatMap

extends Mapper<LongWritable, Text, DBOutputWritable, NullWritable>protected void map(LongWritableid, Text value, Context context)

context.write(new DBOutputWritable(Integer.parseInt(lineArray[1]), lineArray[2], lineArray[3]), NullWritable.get())

Implementation of db output format

public void write(PreparedStatementps) throws SQLException

{

ps.setInt(1, id);

ps.setString(2, name);

ps.setString(3, location);

}