**Big Data And Hadoop**

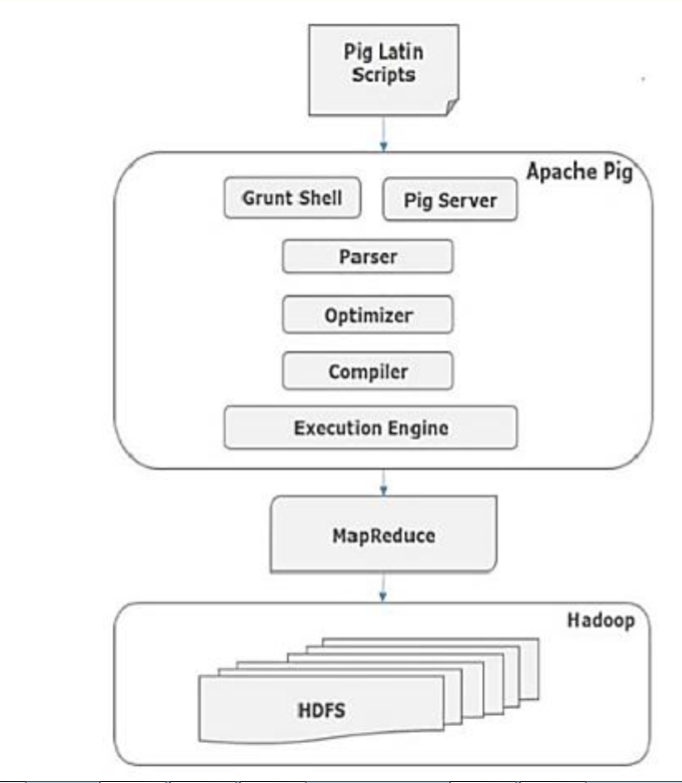
**Session 7 Assignment 1**

Give a brief answers to the questions below:

1. **Why Map-reduce program is needed in Pig Programming?**

**Answer :** Significance of map-reduce program in pig programming is as follows :

1. Pig runs on Hadoop. It makes use of both the Hadoop Distributed File System, HDFS, and Hadoop’s processing system, MapReduce. Its architecture is as follows



1. Map-reduce forms the base of pig. Every pig task executes map-reduce jobs behind the scenes. Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs.
2. In simple terms Map Reduce is low level of programming and Pig is a high-level language for expressing data analysis programs which internally create sequence of Map Reduce Programs.
3. Map-reduce program is needed in pig programming to take advantage of parallel processing as large-scale parallel implementations for map-reduce already exist (e.g., the Hadoop subproject).
4. Pig is written specifically for managing data flow of Map reduce type of jobs.
5. **What are advantages of pig over MapReduce?**

**Answer:** The advantages of pig over map-reduce are as follows :

1. Pig does not demand the users to be hard-core java programmers. Java is not the preferred language for many. Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.
2. The users need not think of the solution in terms of map and reduce functions, which is not the case with map-reduce. Pig provides a high level abstraction for analyzing data.
3. It is quite difficult in MapReduce to perform a Join operation between datasets. Performing a Join operation in Apache Pig is pretty simple.
4. Pig reduces the efforts of the user in terms of writing lengthy programs and debugging them. As lines of code in pig is about 1/20th of that map-reduce, it gets easy for user to write and debug. Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.

Eg : Consider a word-count program

-----------------Map-reduce----------------------

//---------Driver Class-------------

*import org.apache.hadoop.conf.Configuration;*

*import org.apache.hadoop.fs.Path;*

*import org.apache.hadoop.io.IntWritable;*

*import org.apache.hadoop.io.Text;*

*import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;*

*import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;*

*import org.apache.hadoop.mapreduce.Job;*

*import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;*

*import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;*

*public class WordCount*

*{*

*public static void main(String[] args) throws Exception*

*{*

*if (args.length != 2)*

*{*

*System.err.println("Usage: WordCount <input path> <output path>");*

*System.exit(-1);*

*}*

*//Job Related Configurations*

*Configuration conf = new Configuration();*

*Job job = new Job(conf, "My Word Count with combiner");*

*job.setJarByClass(WordCount.class);*

*// Specify the number of reducer to 2*

*job.setNumReduceTasks(2);*

*//Provide paths to pick the input file for the job*

*FileInputFormat.setInputPaths(job, new Path(args[0]));*

*//Provide paths to pick the output file for the job, and delete it if already present*

*Path outputPath = new Path(args[1]);*

*FileOutputFormat.setOutputPath(job, outputPath);*

*outputPath.getFileSystem(conf).delete(outputPath, true);*

*//To set the mapper and reducer of this job*

*job.setMapperClass(WordCountMapper.class);*

*job.setReducerClass(WordCountReducer.class);*

*//Set the combiner*

*job.setCombinerClass(WordCountReducer.class);*

*//set the input and output format class*

*job.setInputFormatClass(TextInputFormat.class);*

*job.setOutputFormatClass(TextOutputFormat.class);*

*//set up the output key and value classes*

*job.setOutputKeyClass(Text.class);*

*job.setOutputValueClass(IntWritable.class);*

*//execute the job*

*System.exit(job.waitForCompletion(true) ? 0 : 1);*

*}*

*}*

*//---------Mapper Class------------*

*import java.io.IOException;*

*import org.apache.hadoop.io.IntWritable;*

*import org.apache.hadoop.io.LongWritable;*

*import org.apache.hadoop.io.Text;*

*import org.apache.hadoop.mapreduce.Mapper;*

*import java.util.\*;*

*public class WordCountMapper*

*extends Mapper<LongWritable, Text, Text, IntWritable> {*

*private final static IntWritable one = new IntWritable(1);*

*private Text word = new Text();*

*@Override*

*public void map(LongWritable key, Text value, Context context)*

*throws IOException, InterruptedException {*

*String line = value.toString();*

*StringTokenizer tokenizer = new StringTokenizer(line);*

*while (tokenizer.hasMoreTokens()) {*

*word.set(tokenizer.nextToken());*

*context.write(word, one);*

*}*

*}*

*}*

*//---------Reducer Class-------------*

*import java.io.IOException;*

*import org.apache.hadoop.io.IntWritable;*

*import org.apache.hadoop.io.Text;*

*import org.apache.hadoop.mapreduce.Reducer;*

*public class WordCountReducer*

*extends Reducer<Text, IntWritable, Text, IntWritable> {*

*@Override*

*public void reduce(Text key, Iterable<IntWritable> values,*

*Context context)*

*throws IOException, InterruptedException {*

*System.out.println("From The Reducer=>"+key) ;*

*int sum = 0;*

*for (IntWritable value : values) {*

*sum+=value.get();*

*}*

*context.write(key, new IntWritable(sum));*

*}*

*}*

*----------------Pig Script----------------*

*A = load '/hadoopdata/wordcount/test.txt';*

*B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;*

*C = group B by word;*

*D = foreach C generate group, COUNT(B);*

*dump D;*

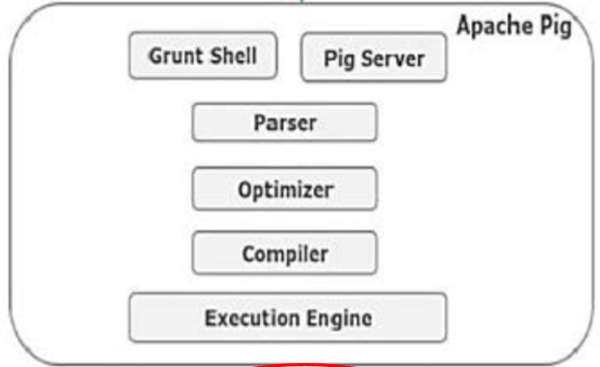
1. Pig uses simple in-built features/operators to reduce development efforts.
2. When using pig, there are fewer java-level bugs to work out.
3. Pig has more opportunities to de automatic optimization of queries. Pig provides inbuilt optimization for MR jobs whereas in map reduce developer needs to take care of optimization.
4. Same pig script will work across versions [most of the part].
5. Pig is simple to learn and use as compared to Map Reduce.

1. **What is pig engine and what is its importance?**

**Answer :** 1. Pig engine is the execution environment to run Pig Latin programs.

2. Pig Engine converts these Pig Latin operators or transformations into a series of MapReduce jobs.

3. Diagrammatically, pig engine can be represented as follows :



Components on pig engine can be explained as follows :

1. Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators. In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

### Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

### Compiler

The compiler compiles the optimized logical plan into a series of MapReduce jobs.

### Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

1. **What are the modes of Pig execution?**

**Answer :** Pig has two execution modes :

1. Local Mode: Local execution in a single JVM, all files are installed and run using local host and file system. Input file has to be kept in local file system in case of local mode. This mode is generally used for testing purpose.

Type the command pig -x local to run Pig in Local Mode.

1. Mapreduce Mode: Distributed execution on a Hadoop cluster, it is the default mode. Here Pig jobs run as a series of MapReduce jobs picking the input and output paths from HDFS. Input file has to be copied in HDFS in case of Map reduce mode. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

Type the command pig or pig –x mapreduce to run Pig in MapReduce Mode.

**Eg :**This example shows how to run Pig in local and mapreduce mode using the pig command.

*/\* local mode \*/*

*$ pig -x local ...*

*/\* mapreduce mode \*/*

*$ pig ...*

*or*

*$ pig -x mapreduce ...*

1. **What is grunt shell in Pig?**

**Answer:** Grunt shell in pig is used to run pig scripts. It is Pig’s interactive shell. It enables users to enter Pig Latin interactively and provides a shell for users to interact with HDFS.

To enter Grunt, invoke Pig with no script or command to run as follows :

pig –x local

or

pig

This will result in the prompt as follows :

grunt>

Grunt provides command-line history and editing, as well as Tab completion. It does not provide filename completion via the Tab key. That is, if you type kil and then press the Tab key, it will complete the command as kill. But if you have a file foo in your local directory and type ls fo, and then hit Tab, it will not complete it as ls foo. This is because the response time from HDFS to connect and find whether the file exists is too slow to be useful.

1. **What are the features of Pig Latin language?**

**Answer :** To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

Its features are as follows :

1. Pig Latin includes operators for many of the traditional data operations (join, sort, filter, etc.)
2. Pig Latin is extensible so that users can develop their own functions for reading, processing, and writing data.
3. Pig Latin script is made up of a series of operations, or transformations, that are  
   applied to the input data to produce output
4. Pig Latin programs can be executed either in Interactive mode through Grunt shell or in Batch mode via Pig Latin Scripts.
5. **Is Pig latin commands case sensitive?**

**Answer :** The names (aliases) of relations and fields are case sensitive. The names of Pig Latin functions are case sensitive. The names of parameters (see Parameter Substitution) and all other Pig Latin keywords are case insensitive.

In the example below, note the following:

1. The names (aliases) of relations A, B, and C are case sensitive.
2. The names (aliases) of fields f1, f2, and f3 are case sensitive.
3. Function names PigStorage and COUNT are case sensitive.
4. Keywords LOAD, USING, AS, GROUP, BY, FOREACH, GENERATE, and DUMP are case insensitive. They can also be written as load, using, as, group, by, etc.
5. In the FOREACH statement, the field in relation B is referred to by positional notation ($0).

grunt> A = LOAD 'data' USING PigStorage() AS (f1:int, f2:int, f3:int);

grunt> B = GROUP A BY f1;

grunt> C = FOREACH B GENERATE COUNT ($0);

grunt> DUMP C;

1. **What is a data flow language?**

**Answer :**

1. Data flow language is a language that manages the flow of data from input source to output store.
2. As part of managing this data flow it moves **data** feeding it to process1, taking output and feeding it to process2, etc.
3. Dataflow programming emphasizes the movement of data and models programs as a series of connections.
4. The core features are preventing execution of subsequent stages if previous stage fails, manages temporary storage of data and most importantly compresses and rearranges processing steps for faster processing. While this can be done for any kind of processing tasks Pig is written specifically for managing data flow of Map reduce type of jobs.
5. In a dataflow language, you have a stream of data which is passed from instruction to instruction to be processed. Conditional execution, jumps and procedure calls route the data to different instructions. They focus on the state of the program and cause operations to occur according to any change in the state.
6. Dataflow programming languages are inherently parallel, because the operations rely on inputs that when met will cause the operation to execute. This means unlike a normal program where one operation is followed by the next operation, in a dataflow program operations will execute as long as the inputs are met and thus there is no set order.
7. Pig Latin is an example of data flow language.

Consider, for example, a simple pipeline, where data from sources users and clicks is to be joined and filtered, and then joined to data from a third source geoinfo and aggregated and finally stored into a table ValuableClicksPerDMA. Pig latin for this will look like :

*Users = load 'users' as (name, age, ipaddr);*

*Clicks = load 'clicks' as (user, url, value);*

*ValuableClicks = filter Clicks by value > 0;*

*UserClicks = join Users by name, ValuableClicks by user;*

*Geoinfo = load 'geoinfo' as (ipaddr, dma);*

*UserGeo = join UserClicks by ipaddr, Geoinfo by ipaddr;*

*ByDMA = group UserGeo by dma;*

*ValuableClicksPerDMA = foreach ByDMA generate group, COUNT(UserGeo);*

*store ValuableClicksPerDMA into 'ValuableClicksPerDMA';*

Pig Latin allows users to describe how data from one or more inputs should be read, processed, and then stored to one or more outputs in parallel. These data flows can be simple linear flows like the word count example given previously. They can also be complex workflows that include points where multiple inputs are joined, and where data is split into multiple streams to be processed by different operators. To be mathematically precise, a Pig Latin script describes a directed acyclic graph (DAG), where the edges are data flows and the nodes are operators that process the data.