Apache Pig

What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with **Hadoop**; we can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using **Apache Pig**, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as **Pig Engine** that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

•Using **Pig Latin**, programmers can perform MapReduce tasks easily without having to type complex codes in Java.

•Apache Pig uses **multi-query approach**, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.

•Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.

•Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features –

•**Rich set of operators** – It provides many operators to perform operations like join, sort, filer, etc.

•**Ease of programming** – Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.

•**Optimization opportunities** – The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.

•**Extensibility** – Using the existing operators, users can develop their own functions to read, process, and write data.

•**UDF's** – Pig provides the facility to create **User-defined Functions** in other programming languages such as Java and invoke or embed them in Pig Scripts.

•**Handles all kinds of data** – Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Apache Pig Vs MapReduce

Listed below are the major differences between Apache Pig and MapReduce.

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

Apache Pig Vs SQL

Listed below are the major differences between Apache Pig and SQL.

Pig	SQL
Pig Latin is a **procedural**language.	SQL is a **declarative**language.
In Apache Pig, **schema**is optional. We can	Schema is mandatory in SQL.

store data without designing a schema (values are stored as $01, $02 etc.)	
The data model in Apache Pig is **nested relational**.	The data model used in SQL **is flat relational**.
Apache Pig provides limited opportunity for **Query optimization**.	There is more opportunity for query optimization in SQL.

In addition to above differences, Apache Pig Latin −

•Allows splits in the pipeline.

•Allows developers to store data anywhere in the pipeline.

•Declares execution plans.

•Provides operators to perform ETL (Extract, Transform, and Load) functions.

Apache Pig Vs Hive

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

Apache Pig	Hive
Apache Pig uses a language called **Pig Latin**. It was originally created at **Yahoo**.	Hive uses a language called **HiveQL**. It was originally created at **Facebook**.
Pig Latin is a data flow language.	HiveQL is a query processing language.
Pig Latin is a procedural language and it fits in pipeline paradigm.	HiveQL is a declarative language.
Apache Pig can handle structured, unstructured, and semi-structured data.	Hive is mostly for structured data.

Applications of Apache Pig

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used −

•To process huge data sources such as web logs.

•To perform data processing for search platforms.

•To process time sensitive data loads.

Apache Pig – History

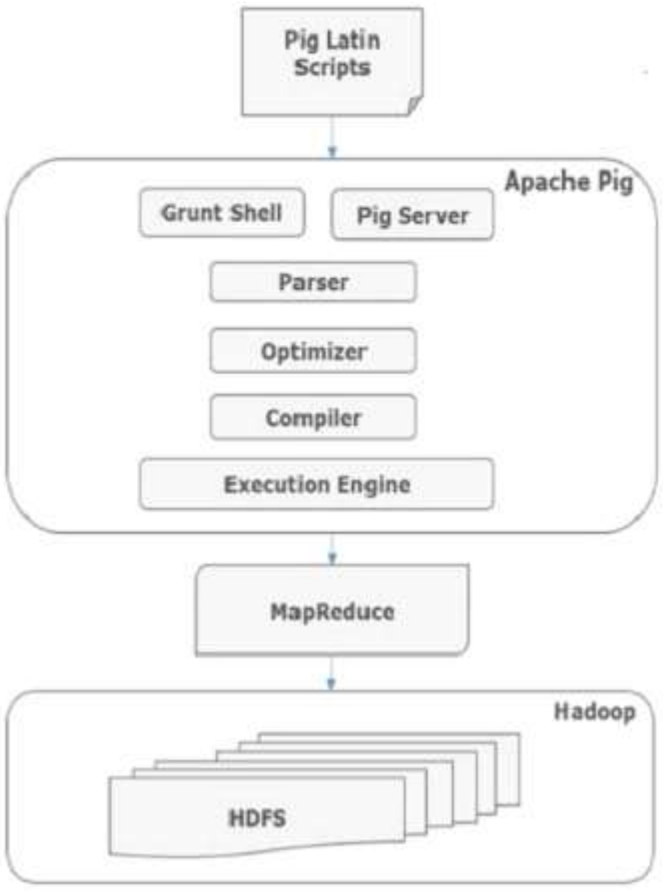
In **2006**, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset. In **2007**, Apache Pig was open sourced via Apache incubator. In **2008**, the first release of Apache Pig came out. In **2010**, Apache Pig graduated as an Apache top-level project.

Apache Pig - Architecture

The language used to analyze data in Hadoop using Pig is 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.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.



Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

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

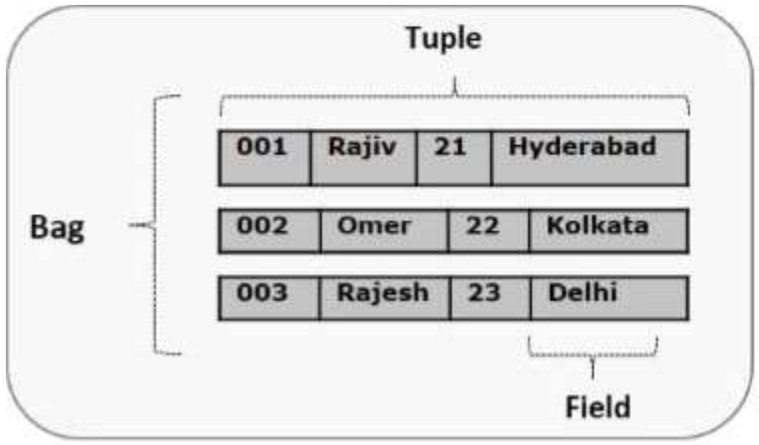
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

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.

Pig Latin Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**. Given below is the diagrammatical representation of Pig Latin's data model.



Atom

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a **field**.

Example – ‘raja’ or ‘30’

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Example – (Raja, 30)

Bag

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by ‘{}’. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example – {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**.

Example – {Raja, 30, **{9848022338, raja@gmail.com,}**}

Map

A map (or data map) is a set of key-value pairs. The **key** needs to be of type chararray and should be unique. The **value** might be of any type. It is represented by ‘[]’

Example – [name#Raja, age#30]

Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

Apache Pig Execution Modes

You can run Apache Pig in two modes, namely, **Local Mode** and **HDFS mode**.

Local Mode

In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

MapReduce Mode

MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. 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.

Apache Pig Execution Mechanisms

Apache Pig scripts can be executed in three ways, namely, interactive mode, batch mode, and embedded mode.

•**Interactive Mode** (Grunt shell) − You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output (using Dump operator).

•**Batch Mode** (Script) − You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with **.pig** extension.

•**Embedded Mode** (UDF) − Apache Pig provides the provision of defining our own functions (**U**ser **D**efined **F**unctions) in programming languages such as Java, and using them in our script.

•Shell Commands

•The Grunt shell of Apache Pig is mainly used to write Pig Latin scripts. Prior to that, we can invoke any shell commands using **sh** and **fs**.

•sh Command

•Using **sh** command, we can invoke any shell commands from the Grunt shell. Using **sh** command from the Grunt shell, we cannot execute the commands that are a part of the shell environment (**ex** − cd).

•Syntax

•Given below is the syntax of **sh** command.

```
•grunt> sh shell command parameters
```

•Example

•We can invoke the **ls** command of Linux shell from the Grunt shell using the **sh** option as shown below. In this example, it lists out the files in the **/pig/bin/** directory.

```
•grunt> sh ls
•
•pig
•pig_1444799121955.log
•pig.cmd
•pig.py
```

•fs Command

•Using the **fs** command, we can invoke any FsShell commands from the Grunt shell.

•Syntax

•Given below is the syntax of **fs** command.

```
•grunt> sh File System command parameters
```

•Example

•We can invoke the ls command of HDFS from the Grunt shell using fs command. In the following example, it lists the files in the HDFS root directory.

```
•grunt> fs –ls
•
•Found 3 items
•drwxrwxrwx   - Hadoop supergroup          0 2015-09-08 14:13 Hbase
•drwxr-xr-x   - Hadoop supergroup          0 2015-09-09 14:52 seqgen_data
•drwxr-xr-x   - Hadoop supergroup          0 2015-09-08 11:30 twitter_data
```

•In the same way, we can invoke all the other file system shell commands from the Grunt shell using the **fs** command.

•Utility Commands

•The Grunt shell provides a set of utility commands. These include utility commands such as **clear, help, history, quit,** and **set**; and commands such as **exec, kill,** and **run** to control Pig from the Grunt shell. Given below is the description of the utility commands provided by the Grunt shell.

•clear Command

•The **clear** command is used to clear the screen of the Grunt shell.

•**Syntax**

•You can clear the screen of the grunt shell using the **clear** command as shown below.

```
•grunt> clear
```

•help Command

•The **help** command gives you a list of Pig commands or Pig properties.

•**Usage**

•You can get a list of Pig commands using the **help** command as shown below.

```
•grunt> help
```

history Command

This command displays a list of statements executed / used so far since the Grunt sell is invoked.

Usage

Assume we have executed three statements since opening the Grunt shell.

```
grunt> customers = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING
PigStorage(',');

grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING
PigStorage(',');

grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING
PigStorage(',');
```

Then, using the **history** command will produce the following output.

```
grunt> history

customers = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING
PigStorage(',');

orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING
PigStorage(',');

student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING
PigStorage(',');
```

set Command

The **set** command is used to show/assign values to keys used in Pig.

Usage

Using this command, you can set values to the following keys.

Key	Description and values
default_parallel	You can set the number of reducers for a map job by passing any whole number as a value to this key.
debug	You can turn off or turn on the debugging freature in Pig by passing on/off to this key.
job.name	You can set the Job name to the required job by passing a string value to this key.
job.priority	You can set the job priority to a job by passing one of the following values to this key − •very_low •low •normal •high •very_high
stream.skippath	For streaming, you can set the path from where the data is not to be transferred, by passing the desired path in the form of a string to this key.

quit Command

You can quit from the Grunt shell using this command.

Usage

Quit from the Grunt shell as shown below.

```
grunt> quit
```

Let us now take a look at the commands using which you can control Apache Pig from the Grunt shell.

exec Command

Using the **exec** command, we can execute Pig scripts from the Grunt shell.

Syntax

Given below is the syntax of the utility command **exec**.

```
grunt> exec [-param param_name = param_value] [-param_file file_name]
[script]
```

Example

Let us assume there is a file named **student.txt** in the **/pig_data/** directory of HDFS with the following content.

Student.txt

```
001,Rajiv,Hyderabad
002,siddarth,Kolkata
003,Rajesh,Delhi
```

And, assume we have a script file named **sample_script.pig** in the **/pig_data/** directory of HDFS with the following content.

Sample_script.pig

```
student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING
PigStorage(',')
   as (id:int,name:chararray,city:chararray);

Dump student;
```

Now, let us execute the above script from the Grunt shell using the **exec** command as shown below.

```
grunt> exec /sample_script.pig
```

Output

The **exec** command executes the script in the **sample_script.pig**. As directed in the script, it loads the **student.txt** file into Pig and gives you the result of the Dump operator displaying the following content.

```
(1,Rajiv,Hyderabad)
(2,siddarth,Kolkata)
(3,Rajesh,Delhi)
```

kill Command

You can kill a job from the Grunt shell using this command.

Syntax

Given below is the syntax of the **kill** command.

```
grunt> kill JobId
```

Example

Suppose there is a running Pig job having id **Id_0055**, you can kill it from the Grunt shell using the **kill** command, as shown below.

```
grunt> kill Id_0055
```

run Command

You can run a Pig script from the Grunt shell using the **run** command

Syntax

Given below is the syntax of the **run** command.

```
grunt> run [–param param_name = param_value] [–param_file file_name] script
```

Example

Let us assume there is a file named **student.txt** in the **/pig_data/** directory of HDFS with the following content.

Student.txt

```
001,Rajiv,Hyderabad
002,siddarth,Kolkata
003,Rajesh,Delhi
```

And, assume we have a script file named **sample_script.pig** in the local filesystem with the following content.

Sample_script.pig

```
student = LOAD 'hdfs://localhost:9000/pig_data/student.txt' USING
   PigStorage(',') as (id:int,name:chararray,city:chararray);
```

Now, let us run the above script from the Grunt shell using the run command as shown below.

```
grunt> run /sample_script.pig
```

You can see the output of the script using the **Dump operator** as shown below.

```
grunt> Dump;

(1,Rajiv,Hyderabad)
(2,siddarth,Kolkata)
(3,Rajesh,Delhi)
```

Note − The difference between **exec** and the **run** command is that if we use **run**, the statements from the script are available in the command history.

Pig Latin – Relational Operations

The following table describes the relational operators of Pig Latin.

Operator	**Description**
Loading and Storing	
LOAD	To Load the data from the file system (local/HDFS) into a relation.
STORE	To save a relation to the file system (local/HDFS).
Filtering	
FILTER	To remove unwanted rows from a relation.
DISTINCT	To remove duplicate rows from a relation.
FOREACH, GENERATE	To generate data transformations based on columns of data.
STREAM	To transform a relation using an external program.
Grouping and Joining	
JOIN	To join two or more relations.
COGROUP	To group the data in two or more relations.
GROUP	To group the data in a single relation.
CROSS	To create the cross product of two or more relations.
Sorting	
ORDER	To arrange a relation in a sorted order based on one or more fields (ascending or descending).
LIMIT	To get a limited number of tuples from a relation.
Combining and Splitting	
UNION	To combine two or more relations into a single relation.
SPLIT	To split a single relation into two or more relations.
Diagnostic Operators	
DUMP	To print the contents of a relation on the console.
DESCRIBE	To describe the schema of a relation.
EXPLAIN	To view the logical, physical, or MapReduce execution plans to compute a relation.
ILLUSTRATE	To view the step-by-step execution of a series of statements.