Chapter 10

Introduction to Pig

What is Pig?

What is Pig?

Apache Pig is a platform for data analysis.

It is an alternative to Map Reduce Programming.

Pig was developed as a research project at yahoo.

Features of Pig

Features of Pig

- It provides an **engine** for executing **data flows** (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
- It provides a language called "**Pig Latin**" to express data flows.
- Pig Latin contains operators for many of the traditional data operations such as join, filter, sort, etc.
- It allows users to develop their own functions (User Defined Functions) for reading, processing, and writing data.

The Anatomy of Pig

The Anatomy of Pig

The main components of Pig are as follows:

- Data flow language (**Pig Latin**).
- Interactive shell where you can type Pig Latin statements (**Grunt**).
- Pig interpreter and execution engine.

Pig Latin

Pig Latin Script

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

A = filter A by gpa >4;

A = foreach A generate UPPER (name);

STORE A INTO ‘myreport’;
```

•**Stores the final output** into an HDFS directory or local file system path called 'myreport'.
•The result will be stored in **HDFS** under /user/hadoop/myreport/part-r-00000.

Pig Interpreter/Execution Engine

- Processes and parses pig Latin
- Checks data types
- Performs optimization
- Creates MapReduce jobs
- Submits jobs to Hadoop
- Monitor progress

Pig on Hadoop

Pig on Hadoop

- Pig runs on Hadoop.
- Pig uses both Hadoop Distributed File System and MapReduce Programming.
- By default, Pig reads input files from HDFS. Pig stores the intermediate data (data produced by MapReduce jobs) and the output in HDFS.
- However, Pig can also read input from and place output to other sources.
- Pig Supports the following:
  - ✓ HDFS commands
  - ✓ UNIX shell commands
  - ✓ Relational operators
  - ✓ Positional parameters
  - ✓ Common mathematical functions
  - ✓ Custom functions
  - ✓ Complex data structures

Positional parameters

Positional parameters in Pig allow you to write **dynamic and reusable scripts** by using **placeholders** for values. These placeholders can be replaced at runtime when executing the script.

```
A = LOAD '$input_path' USING PigStorage(',') AS (id:int, name:chararray, marks:int);
B = FILTER A BY marks > $threshold;
DUMP B;
```

Executing the Script with Parameters
Run the script by passing values using -param:

```
pig -param input_path='/data/students.csv' -param threshold=50 script.pig
```

- $input_path is replaced with '/data/students.csv'.
- $threshold is replaced with 50.

Reusability – Write one script and use different values dynamically.
Flexibility – Easily change input data paths, filters, or other parameters.
Maintainability – No need to hard-code values in scripts.

Pig Philosophy

Apache Pig Vs 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.
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.

ApachePig vs Hive

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.

What Does "Unstructured Data" Mean in Pig?

In Big Data processing, unstructured data generally means text-heavy data without a predefined schema, such as:

Log files (server logs, clickstream data)
JSON, XML files (nested, irregular structures)
Social media posts, emails, chat messages

While this data may not fit into traditional row-column databases, it is still text-based and can be processed by Pig.

Aspect	Declarative Language	Procedural Language
Focus	What needs to be done	How to perform tasks step-by-step
Control Flow	Abstracted (handled by the system)	Explicitly controlled by the programmer
Execution	Describes desired results	Describes the procedure to get results
Examples	SQL, Hive, HTML, Prolog	C, Java, Python, C++, Pig Latin
Ease of Use	Easier to write for data processing	Requires detailed instructions
Parallelism	Implicit (handled by the system)	Must be managed manually
Use Case	Data querying, configuration, and transformation	Algorithm development, system programming
Code Length	Typically shorter and more readable	Often longer with more lines of code

Declarative (SQL): SELECT name FROM students WHERE grade > 80;
Specify what we want—names of students with grades above 80.
Procedural (Python):

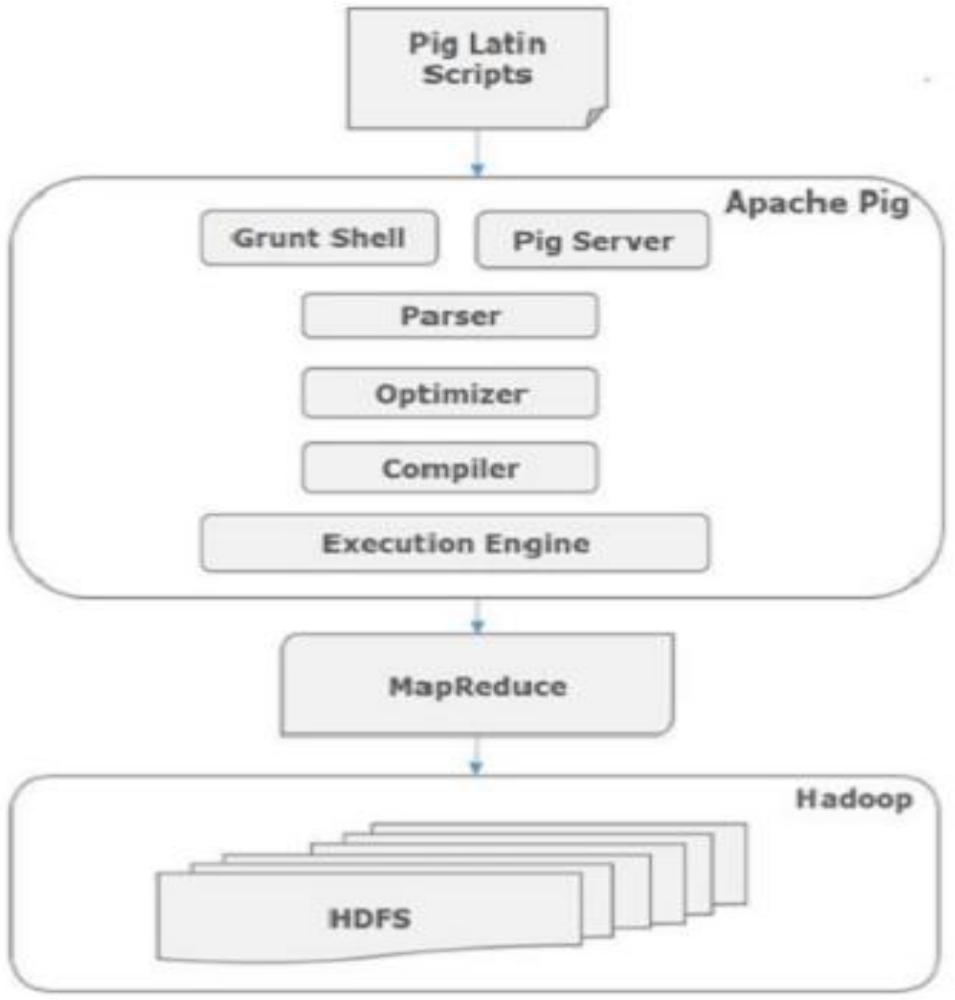
```
result = []
for student in students:
   if student['grade'] > 80:
      result.append(student['name’])
```

specify how to find the names—loop, check condition, and store results.

Pig, Hive, and Hadoop are all used for big data processing, but they serve different purposes.

Feature	Apache Pig	Apache Hive	Hadoop (MapReduce, HDFS, YARN)
Best For	Data transformation (ETL), semi-structured data	SQL-based querying on structured data	Low-level custom big data processing
Ease of Use	Script-based (Pig Latin)	SQL-like queries (HiveQL)	Java-based (MapReduce)
Performance	Faster than raw MapReduce, but slower than Hive on Tez	Optimized for large datasets	Fine-tuned performance, requires Java coding
Data Type	Structured & large text-based unstructured	Structured (tables, schema-based)	Structured & semi-structured
Integration	Works with HDFS, Tez, Spark	Works with HDFS, Tez, Spark	Base layer for all big data tools

Pig Architecture



PARSER:

- 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.
- The DAG structure ensures that there are no **cycles**, meaning no operator can depend on itself indirectly through a chain of operations. This is important for ensuring the program has a valid, executable order of operations.

Optimizer

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

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 Architecture

Pig Server:
•Allows integration with applications via Java or other programming interfaces.
•Used when Pig is embedded in Java programs.

Parsing:
•The **Parser** checks Pig Latin scripts for **syntax errors**.
•Converts the script into a **Logical Plan**, representing the sequence of operations (e.g., load → filter → group).

Optimizer:
•Optimizes the **Logical Plan** without changing its semantics.
•Techniques:
✓ Reordering operations for efficiency.
✓ Combining certain operations to reduce MapReduce jobs.
✓ Removing redundant steps.

Compiler:
•Transforms the optimized logical plan into a **Physical Plan**.
•Converts high-level Pig instructions into **low-level MapReduce jobs** for Hadoop.

Pig Architecture

Execution Engine:
•Executes the physical plan by submitting MapReduce jobs to Hadoop.
•Monitors job progress and handles task scheduling.

MapReduce Layer:
•Acts as the execution backbone.
•Processes data using the **MapReduce paradigm**:

- **Map:** Data is processed in parallel across nodes.
- **Reduce:** Aggregates or transforms the mapped data.

Pig Latin Statements

Pig Latin Statements are generally ordered as follows:

1. **LOAD** statement that reads data from the file system.
2. Series of statements to perform transformations.
3. **DUMP** or **STORE** to display/store result.

```
A = load 'student' (rollno, name, gpa);

A = filter A by gpa > 4.0;


A = foreach A generate UPPER (name);


STORE A INTO ‘myreport’
```

Pig Latin Identifiers

1. Identifiers are names assigned to fields or other data structures
2. It should begin with a letter and should be followed by only letters, numbers, and underscores.

Valid Identifier	Y	A1	A1_2014	Sample
Invalid Identifier	5	Sales$	Sales%	_Sales

Pig Latin Comments

In Pig Latin two types of comments are supported:

1. Single line comments that begin with “—”.
2. Multiline comments that begin with “/* and end with */”.

Pig Latin case sensitivity

Operators in Pig Latin

Arithmetic	Comparison	Null	Boolean
+	= =	IS NULL	AND
-	! =	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

Data Types in Pig Latin

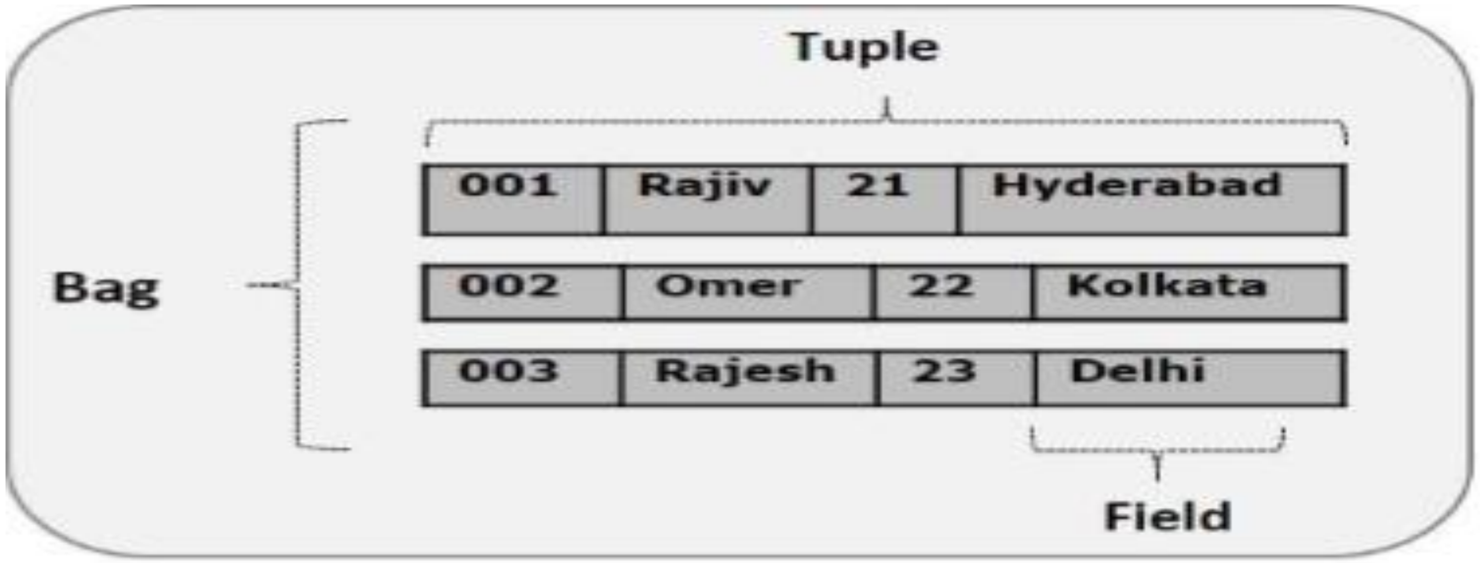
Simple Data Types

Name	Description
int	Whole numbers
long	Large whole numbers
float	Decimals
double	Very precise decimals
chararray	Text strings
bytearray	Raw bytes
datetime	Datetime
boolean	true or false

Complex Data Types

Name	Description
Tuple	An ordered set of fields. Example: (2,3)
Bag	A collection of tuples. Example: {(2,3),(7,5)}
map	key, value pair (open # Apache)

Complex Data type



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.

Complex data type: 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)}

Complex data structures

Pig supports Tuples, Bags, and Maps for handling complex data.

A Bag in Pig is a collection of tuples. It is similar to a list of rows in a table.
Key Properties of Bags:
✓ Unordered (no specific sequence).
✓ Can contain duplicate tuples.
✓ Can be nested (bags inside bags).

```
A = LOAD 'students.txt' AS (id:int, name:chararray, scores:bag{tuple(subject:chararray, marks:int)});
```

```
1,John,{(Math,85),(Science,90)}
2,Alice,{(Math,78),(English,88)}
```

When to Use Bags?
✓ When handling nested data.
✓ When performing grouping operations.
✓ When dealing with multi-valued attributes (e.g., a student with multiple test scores).

Complex data type-MAP and Relation

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.

Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered

Running Pig

Pig can run in two ways:

1. Interactive Mode: By invoking **GRUNT** shell

2. Batch Mode: Write Pig Latin in File save it with .pig extension

Execution Modes of Pig

You can execute pig in two modes:

1. Local Mode.

   Keep your files in local file system.

   pig -x Local file name

2. Map reduce

   Need access Hadoop cluster to read/write file.

   Default mode of pig

   Pig File name

HDFS commands

All HDFS commands can run in PIG

Shell command :

```
grunt> sh ls
```

HDFS command

```
Grunt> fs –ls
```

Commands:

Quit
Help
clear
exec
run

Relational Operators

Filter

Find the tuples of those student where the GPA is greater than 4.0.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = filter A by gpa > 4.0;
DUMP B;
```

FOREACH

Display the name of all students in uppercase.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = foreach A generate UPPER(name);

DUMP B;
```

Group

Group tuples of students based on their GPA.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = GROUP A BY gpa;


DUMP B;
```

COGROUP

Input file:data1.tsv

A	cat
B	dog
D	turtle
F	cat
G	cat
E	fish

Input file:data2.tsv

xx	A
yy	A
vv	B
HH	D
JJ	E

```
grunt> petdata = load '/home/hduser/Desktop/data1.tsv' as (owner:chararray,pet:chararray);
grunt> dump petdata;

grunt> frienddata = load '/home/hduser/Desktop/data2.tsv' as(f1:chararray,f2:chararray);
grunt> dump frienddata;

grunt> y = cogroup petdata by owner, frienddata by f2;
grunt> dump y;

(A,{(A,cat)},{(xx,A),(yy,A)})
(B,{(B,dog)},{(vv,B)})
```

Distinct

To remove duplicate tuples of students.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = DISTINCT A;

DUMP B;
```

Limit

Limit the number of output tuples

Display the first three tuples from “student” relation

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = LIMIT A 3;

DUMPB;
```

ORDER BY

Used to sort relation

Display the names of the students in Ascending order.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = ORDER A BY name;

DUMP B;
```

Join

To join two relations namely, “student” and “department” based on the values contained in the “rollno” column.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int,deptname:chararray);

C = JOIN A BY rollno, B BY rollno;

DUMP C;

DUMP B;
```

UNION

It is used to merge the contents of two relations. Merge datasets **vertically** without any condition. Combines two or more relations with the same schema into one.

To merge two relations “student” and “department”

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int,deptname:chararray);

C = UNION A,B;

STORE C INTO ‘/pigdemos/uniondemo’;

DUMPC;
```

Split

To partition a relation based on the GPAs acquired by the students.

- GPA = 4.0, place it into relation X.
- GPA is < 4.0, place it into relation Y.

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

SPLIT A INTO X IF gpa==4.0, Y IF gpa<=4.0;

DUMP X;
```

SAMPLE

Used to select random sample of data based on the specified sample size. Extracts a random subset of data

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

B = SAMPLE A 0.01; -- 1% random sample

DUMPB;
```

Operator	Purpose	Example Use Case
JOIN	Combine datasets by key	Student info with marks data
SPLIT	Divide data into subsets	Separate pass and fail records
UNION	Merge datasets of same schema	Combine class A and B records
SAMPLE	Randomly select data subset	Select 10% of data for testing

Eval Function

Avg

To calculate the average marks for each student.

```
A = load '/pigdemo/student.csv' USING PigStorage (‘,’) as
(studname:chararray,marks:int);

B = GROUP A BY studname;

C = FOREACH B GENERATE A.studname,AVG(A.marks);

DUMP C;
```

Pig Storage is a built-in function of Pig, and one of the most common functions used to load and store data in pig scripts. Pig Storage can be used to parse text data with an arbitrary delimiter, or to output data in a delimited format.

Max

To calculate the maximum marks for each student.

```
A = load '/pigdemo/student.csv' USING PigStorage (‘,’) as (studname:chararray,marks:int);

B = GROUPA BY studname;


C = FOREACH B GENERATE A.studname, MAX(A.marks);

DUMPC;
```

COUNT

used to count the number of elements in a bag

```
A = load '/pigdemo/student.csv' USING PigStorage (‘,’) as
(studname:chararray,marks:int);

B = GROUPA BY studname;

C = FOREACH B GENERATE A.studname, COUNT(A);

DUMP C;
```

COMPLEX DATA TYPES

TUPLE

Tuple is an ordered collections of fields

INPUT:

```
(x,12)      (y,13)
(z,7)       (f,5)
(s,8)       (sc,12)
```

```
A=load '/root/pigdemos/studentdata.tsv' AS

(t1:tuple(t1a:chararray,t1b:int),t2:tuple(t2a:chararray,t2b:int));

B= FOREACH A GENERATE t1.t1a,t1.t1b,t2.$0,t2.$1;

DUMP B;
```

Note: By default, Pig uses a **tab** (\t) delimiter for .tsv files.

Output:

```
(x,12,y,13)
(z,7,f,5)
(s,8,sc,12)
```

Map

MAP represents a key/value pair.

To depict the complex data type “map”.

```
John      [city#Bangalore]
Jack      [city#Pune]
James     [city#Chennai]
```

```
A = load '/root/pigdemos/studentcity.tsv' Using PigStorage as
(studname:chararray,m:map[chararray]);

 B = foreach A generate m#'city' as CityName:chararray;


  DUMP B
```

Piggy Bank

Piggy Bank

Piggybank **is Pig's repository of user-contributed functions. Piggybank functions are distributed as part of the Pig distribution, but they are not built in. You must register the Piggybank JAR to use them.**

```
register '/root/pigdemos/piggybank-0.12.0.jar';

A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

upper = foreach A generate
org.apache.pig.piggybank.evaluation.string.UPPER(name);

DUMP upper;
```

USER DEFINED FUNCTIONS -UDF

To convert Names into **upper case**

```
package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappaedIOException;

public class UPPER extends EvalFunc<string>
{
public String exec(Tuple input) throws IOException{
if(input==null || input.size() ==0)
return null;
try{String str=(string input.get(0);
return str.toUppercase();
}
catch(Exception e){
Throw WrappedIOExceptio.wrap("Caught exception",e):
}}}
```

UDF

Convert the java class into jar

```
register /root/pigdemos/myudfs.jar;

 A= load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

 B=FOREACH A GENERATE myudfs.UPPER(name);

 DUMP B;
```

Parameter substitution

Pig allows to pass parameters at run time

```
A = load ‘$student' as (rollno:int, name:chararray, gpa:float);

DUMP A;
```

Execute:

```
pig –param student =/pigdemo/student.tsv  parameterdemo.pig
```

Diagnostic operators

It returns the schema of a relation

DESCRIBE

```
A= load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

DESCRIBE A;
```

Word count example in PIG

```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);

words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;

grouped = GROUP words BY word;

wordcount = FOREACH grouped GENERATE group, COUNT(words);

DUMP wordcount;
```

Tokenize and Flatten

Tokenize :

Splits the line into a field for each word.

Flatten:

will take the collection of records returned by TOKENZIE and produce a separate record for each one, calling the single field in the record word.

When to use Pig?

When to use Pig?

Pig can be used in the following situations:

1. When data loads are time sensitive.
2. When processing various data sources.
3. When analytical insights are required through sampling.

When NOT to use Pig?

Pig should not be used in the following situations:

1. When data is completely unstructured such as video, text, and audio.
2. When there is a time constraint because Pig is slower than MapReduce jobs.

Pig Vs. Hive

Pig Vs. Hive

Features	Pig	Hive
Used By	Programmers and Researchers	Analyst
Used For	Programming	Reporting
Language	Procedural data flow language	SQL Like
Suitable For	Semi - Structured	Structured
Schema / Types	Explicit	Implicit
UDF Support	YES	YES
Join / Order / Sort	YES	YES
DFS Direct Access	YES (Implicit)	YES (Explicit)
Web Interface	YES	NO
Partitions	YES	No
Shell	YES	YES

Answer a few quick questions …

Fill in the blanks

1. Pig is a____________language.
2. In Pig,____________is used to specify data flow.
3. Pig provides an____________to execute data flow.
4. ____________,_____________are execution modes of Pig.
5. Pig is used in____________process.

References …

Further Readings

- http://pig.apache.org/docs/r0.12.0/index.html
- http://www.edureka.co/blog/introduction-to-pig/

Thank you