Unit 5

Pig: Hadoop Programming Made Easier Admiring the Pig Architecture, going with the Pig Latin Application Flow, working through the ABCs of Pig Latin, Evaluating Local and Distributed Modes of Running Pig Scripts, Checking out the Pig Script Interfaces, Scripting with Pig Latin

- Java MapReduce programs and the Hadoop Distributed File System (HDFS) provide you with a powerful distributed computing framework, but they relying on them limits the use of Hadoop to Java programmers who can think in Map and Reduce terms when writing programs.
- More developers, data analysts, data scientists, and other people could gain advantage of Hadoop if they had a way to understand the power of Map and Reduce while hiding some of the Map and Reduce complexities.
- Hive and Pig hide the messy details of MapReduce so that a programmer can concentrate on the important work.
- Hive, for example, provides a limited SQL-like capability that runs over MapReduce, thus making said MapReduce more approachable for SQL developers.
- Hive also provides a *declarative* query language (the SQL-like HiveQL), which allows you to focus on *which* operation you need to carry out versus *how* it is carried out
- SQL is the common accepted language for querying structured data, some developers still prefer writing imperative scripts — scripts that define a set of operations that change the state of the data — and also want to have more data processing flexibility than what SQL or HiveQL provide.
- Engineers at Yahoo! Research to come up with a product meant to fulfil that need — and so Pig was born
- Pig's claim to a programming tool attempting to have:
    - a declarative query language inspired by SQL and
    - a low-level procedural programming language that can generate MapReduce code.
- This simplifies technical knowledge needed to exploit the power of Hadoop.
- Pig was initially developed at Yahoo! in 2006 as part of a research project, in 2007 Pig officially became an Apache project
- The Pig programming language is designed to handle any kind of data tossed its way — structured, semi- structured, unstructured data
- According to the Apache Pig philosophy, pigs eat anything, live anywhere, are domesticated and can fly to boot.
- Pig is a parallel data processing programming language and can work with any particular parallel framework.

- Pig is smart in data processing because there is an optimizer that figures out how to do the hard work of figuring out how to get the data quickly.

Admiring the Pig Architecture

- Pig is made up of two components:
- The language itself: the programming language for Pig is known as Pig Latin, a high-level language that allows you to write data processing and analysis programs.
- The Pig Latin compiler: The Pig Latin compiler converts the Pig Latin code into executable code. The executable code is either in the form of MapReduce jobs or to run the Pig code on a single node.
- The sequence of MapReduce programs enables Pig programs to do data processing and analysis in parallel, leveraging Hadoop MapReduce and HDFS.
- Running the Pig job in the virtual Hadoop instance is a useful strategy for testing your Pig scripts.
- Pig programs can run on MapReduce v1 or MapReduce v2 without any code changes, regardless of what than MapReduce.
- Pig scripts can also run using the Tez API instead. Apache Tez provides a more efficient execution framework than MapReduce.

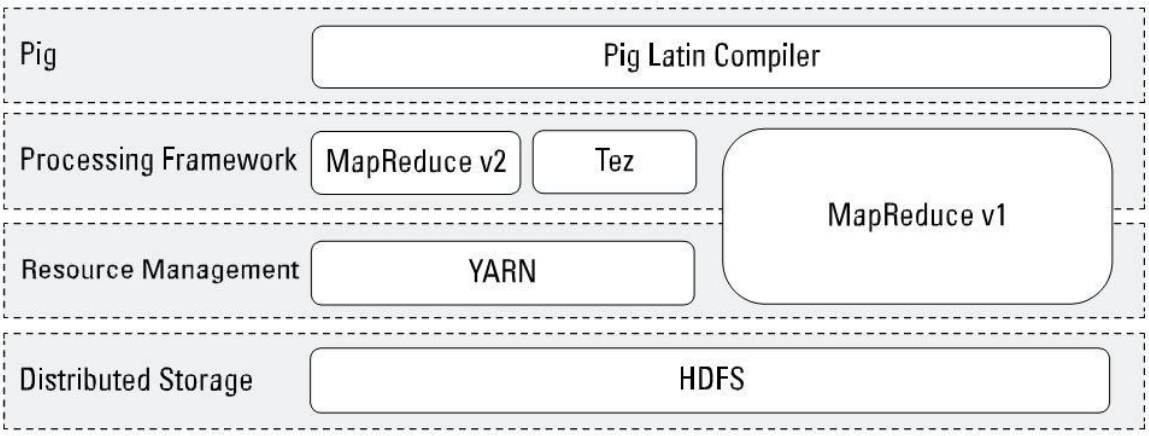

Figure 8-1: Pig architecture.

Going with the Pig Latin Application Flow

- Pig Latin is a dataflow language, where you define a data stream and a series of transformations that are applied to the data as it flows through your application
- A *control flow* language (like C or Java), where you write a series of instructions. In control flow languages, we use constructs like loops and conditional logic (like an ifstatement).
- To realize working with pig is significantly easier than MapReduce consider the following example

Listing 8-1: Sample Pig Code to illustrate the data processing dataflow

```
A = LOAD 'data_file.txt';
...
B = GROUP ... ;
...
C= FILTER ...;
...
DUMP B;
..
STORE C INTO 'Results';
```

The basic flow of a Pig program is:

- 1. Load: First load (LOAD) the data we want to manipulate, that data is stored in HDFS or local file system. For a Pig program to access the data, you first tell Pig what file or files to use. For that task, you use the LOAD 'data_file' command. Here, 'data_file' can specify either an HDFS file or a directory. If a directory is specified, all files in that directory are loaded into the program
- If the data is stored in a file format that isn't natively accessible to Pig, you can optionally add the USING function to the LOAD statement to specify a user-defined function that can read in (and interpret) the data
- 2.Transform: We run the data through a set of transformations that are translated into a set of Map and Reduce tasks.
- The transformation logic is where all the data manipulation happens. Here, you can FILTER out rows that aren't of interest, JOIN two sets of data files, GROUP data to build aggregations, ORDER results, and do much, much more
- 3. Dump: Finally, you dump (DUMP) the results to the screen or

- Store (STORE) the results in a file somewhere.

Working through the ABCs of Pig Latin

- Pig Latin is the language for Pig programs.
- Pig translates the Pig Latin script into MapReduce jobs that can be executed within Hadoop cluster.
- Pig Latin development team followed three key design principles:

Keep it simple.

- Pig Latin is an abstraction for MapReduce that simplifies the creation of parallel programs on the Hadoop cluster for data flows and analysis.
- Complex tasks may require a series of interrelated data transformations — such series are encoded as *data flow sequences.*
- Writing Pig Latin scripts instead of Java MapReduce programs makes these programs easier to write, understand, and maintain because
- a) you don't have to write the job in Java,
- b) you don't have to think in terms of MapReduce, and
- c) you don't need to come up with custom code to support rich data types.
- Pig Latin provides a simpler language to exploit your Hadoop cluster.

Make it smart.

- Pig Latin Compiler transform a Pig Latin program into a series of Java MapReduce jobs.
- The compiler can optimize the execution of these Java MapReduce jobs automatically, allowing the user to focus on semantics rather than on how to optimize and access the data.
- For example, SQL is set up as a declarative query that you use to access structured data stored in an RDBMS. The RDBMS engine first translates the query to a data access method and then looks at the statistics and generates a series of data access approaches. The cost-based optimizer chooses the most efficient approach for execution.

Don't limit development. Make Pig extensible so that developers can add functions to address their particular business problems.

- ✓ Traditional RDBMS data warehouses make use of the ETL data processing pattern, where you **extract** data from outside sources, **transform** it to fit your operational needs, and then **load** it into the end target, whether it's an operational data store, a data warehouse, or another variant of database.

- ✓ With big data, the language for Pig data flows goes with ELT instead: ***E*xtract** the data from your various sources, ***l*oad** it into HDFS, and then ***t*ransform** it as necessary to prepare the data for further analysis.

Uncovering Pig Latin structures:

For example, consider the following pig scripts that perform the task of calculating the total number of flights flown by every carrier

Listing 8-2: Pig script calculating the total miles flown

Listing 8-2: Pig script calculating the total miles flown

```
records = LOAD '2013 subset.csv' USING PigStorage(',') AS
          (Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDep
          Time,ArrTime,CRSArrTime,UniqueCarrier,FlightNum
          ,TailNum,ActualElapsedTime,CRSElapsedTime,AirTi
          me,ArrDelay,DepDelay,Origin,Dest,Distance:int,T
          axiIn,TaxiOut,Cancelled,CancellationCode,Divert
          ed,CarrierDelay,WeatherDelay,NASDelay,SecurityD
          elay,LateAircraftDelay);

milage_recs = GROUP records ALL;
tot_miles = FOREACH milage_recs GENERATE
          SUM(records.Distance);

DUMP tot_miles;
```

Pig Scripts consider the following principles:

Most Pig scripts start with the LOADstatement to read data from HDFS.

- ✓ In this case, we're loading data from a .csvfile. Pig has a data model it uses, so next we need to map the file's data model to the Pig data mode. This is accomplished with the help of the USING statement. We then specify that it is a comma-delimited file with the PigStorage (',') statement followed by the AS statement defining the name of each of the columns.

Aggregations are commonly used in Pig to summarize data sets.

- ✓ The GROUPstatement is used to aggregate the records into a single record mileage_recs. The ALL statement is used to aggregate all tuples into a single group. Note that some statements — including the following SUM statement — requires a preceding GROUP ALL statement for global sums.

FOREACH . . . GENERATEstatements are used here to transform columns data.

- ✓ In this case, we want to count the miles travelled in the records_Distance column. The SUM statement computes the sum of the record_Distance

column into a single-column collection total_miles.

The DUMPoperator is used to execute the Pig Latin statement and dis- play the results on the screen.

- ✓ DUMPis used in interactive mode, which means that the statements are executable immediately and the results are not saved. Typically, you will either use the DUMP or STORE opera- tors at the end of your Pig script.

Looking at Pig data types and syntax:

Pig Latin has these four types in its data model:

Atom: An *atom is* any single value, such as a string or a number — 'Diego', for example. Pig's atomic values are scalar types that appear in most programming languages — int, long, float, double, chararray, and bytearray, for example. See Figure 8-2 to see sample atom types.

Tuple: A *tuple* is a record that consists of a sequence of fields. Each field can be of any type — 'Diego', 'Gomez', or 6, for example. Think of a tuple as a row in a table

Bag: A *bag* is a collection of non-unique tuples. The schema of the bag is flexible — each tuple in the collection can contain an arbitrary number of fields, and each field can be of any type.

Map: A map is a collection of key value pairs. Any type can be stored in the value, and the key needs to be unique. The key of a map must be a chararrayand the value can be of any type.

Figure 8-2 offers some fine examples of Tuple, Bag, and Map data types, as well.

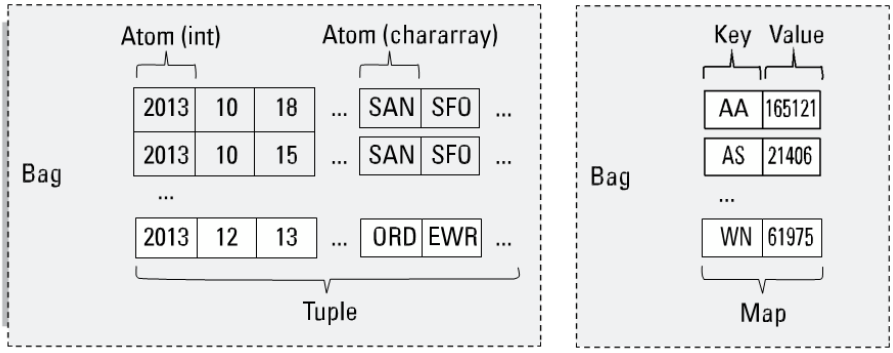

Figure 8-2: Sample Pig Data Types

In a Hadoop context, *accessing data* means allowing developers to load, store, and s t r e a m data, whereas *transforming* data means taking advantage of Pig's ability to group, join, combine, split, filter, and sort data. Table 8-1 gives an overview of the operators associated with each operation

Table 8-1	Pig Latin Operators	
Operation	***Operator***	***Explanation***
Data Access	LOAD/STORE	Read and Write data to file system
	DUMP	Write output to standard output (stdout)
	STREAM	Send all records through external binary
	FOREACH	Apply expression to each record and output one or more records
	FILTER	Apply predicate and remove records that don't meet condition
	GROUP/ COGROUP	Aggregate records with the same key from one or more inputs
	JOIN	Join two or more records based on a condition
Transformations	CROSS	Cartesian product of two or more inputs
	ORDER	Sort records based on key
	DISTINCT	Remove duplicate records
	UNION	Merge two data sets
	SPLIT	Divide data into two or more bags based on predicate
	LIMIT	subset the number of records

Table 8-2	Operators for Debugging and Troubleshooting	
Operation	***Operator***	***Description***
Debug	DESCRIBE	Return the schema of a relation.
	DUMP	Dump the contents of a relation to the screen.
	EXPLAIN	Display the MapReduce execution plans.

The LOADoperator operates on the principle of *lazy evaluation,* also referred to as *call-by-need.* Now lazy doesn't sound particularly praiseworthy, but all it means is that you delay the evaluation of an expression until you really need it. In the context of our Pig example, that means that after the LOADstatement is executed, no data is moved — nothing gets shunted around — until a statement to write data is encountered. You can have a Pig script that is a page long filled with complex transformations, but nothing gets executed until the DUMPor STOREstatement is encountered.

Evaluating Local and Distributed Modes of Running Pig scripts:

Pig has two modes for running scripts, as shown in Figure 8-3:

Local mode: All scripts are run on a single machine without requiring Hadoop MapReduce and HDFS. This can be useful for developing and testing Pig logic. If you're using a small set of data to developed or test your code, then local mode could be faster than going through the MapReduce infrastructure.

- ✓ Local mode doesn't require Hadoop. When you run in Local mode, the Pig

program runs in the context of a local Java Virtual Machine, and data access is via the local file system of a single machine. Local mode is actually a local simulation of MapReduce in Hadoop's LocalJobRunner class.

- ✓ **MapReduce mode (also known as Hadoop mode):** Pig is executed on the Hadoop cluster. In this case, the Pig script gets converted into a series of MapReduce jobs that are then run on the Hadoop cluster.

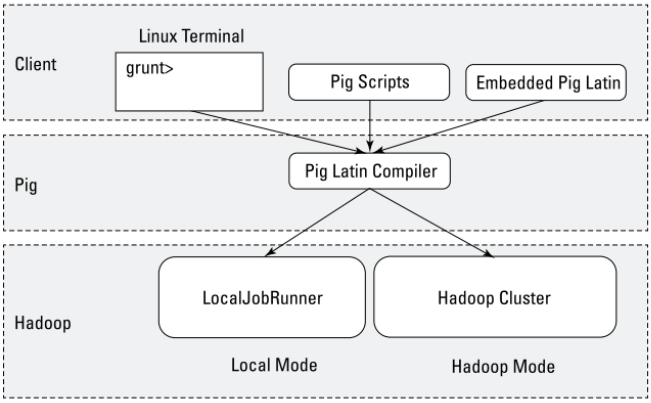

Figure 8-3: Pig modes

Checking Out the Pig Script Interfaces

Pig programs can be packaged in three different ways:

Script: This method is nothing more than a file containing Pig Latin commands, identified by the .pigsuffix (FlightData.pig, for example). Ending your Pig program with the .pigextension is a convention but not required. The commands are interpreted by the Pig Latin compiler and executed in the order determined by the Pig optimizer.

Grunt: Grunt acts as a command interpreter where you can interactively enter Pig Latin at the Grunt command line and immediately see the response. This method is helpful for prototyping during initial development and with what-if scenarios.

Embedded: Pig Latin statements can be executed within Java, Python, or JavaScript programs.

To specify whether a script or Grunt shell is executed locally or in Hadoop mode just specify it in the –xflag to the pigcommand. The following is an example of how you'd specify running your Pig script in local mode:

```
pig -x local milesPerCarrier.pig
```

Here's how you'd run the Pig script in Hadoop mode, which is the default if you don't specify the flag:

```
pig -x mapreduce milesPerCarrier.pig
```

By default, when you specify the pigcommand without any parameters, it starts the Grunt shell in Hadoop mode. If you want to start the Grunt shell in local mode, just add the –x localflag to the command. Here is an example:

```
pig -x local
```

Scripting with Pig Latin

- ✓ Hadoop is a rich and quickly evolving ecosystem with a growing set of new applications, Pig is designed to be extensible via *user-defined functions, also* known as UDFs. UDFs can be written in a number of programming languages, including Java, Python, and JavaScript.
- ✓ Some of the Pig UDFs that are part of these repositories are LOAD/STOREfunctions (XML, for example), date time functions, text, math, and stats functions.
- ✓ Pig can also be embedded in host languages such as Java, Python, and JavaScript, which allows you to integrate Pig with your existing applications. It also helps overcome limitations in the Pig language.
- ✓ One of the most commonly referenced limitations is that Pig doesn't support control flow statements: if/else, while loop, for loop, and conditionstatements.
- ✓ Pig natively supports data flow, but needs to be embedded within another language to provide control flow. There are trade-offs, however of embedding Pig in a control-flow language. For example, if a Pig statement is embedded in a loop, every time the loop iterates and runs the Pig statement, this causes a separate MapReduce job to run.