INTRODUCTION TO APACHE PIG

Apache Pig 101

Apache Pig, developed at Yahoo, was written to make it easier to work with Hadoop. Hadoop was developed by Google. Pig lets programmers work with Hadoop datasets using a syntax that is similar to SQL. Without Pig, programmers most commonly would use Java, the language Hadoop is written in. But Java code is inherently wordy. It would be nicer to have an easier and much shorter way to do Hadoop MapReduce operations. That is what Pig does.

But Pig is not exactly SQL. SQL programmers, in fact, will find some of its data structures a little strange.

Data in Pig is represented in data structures called **tuples**, with all the other Pig data structures being some variation on that. In its most basic form, a tuple is a comma delimited set of values:

```
(1,2,3,4,5)
(1,6,7,8,9)
```

And when you join two tuples, in this case on the first element, it represents the data like this:

```
(1,2,3,4,5,1,6,7,8,9)
```

Or this:

```
(1, {(1,2,3,4,5), (1,2,3,4,5)})
```

In the first case Pig has joined all the elements of two tuples into one. In the second it has put the join criteria in the first element and created a bag in the second. A **bag** is a collection of tuples. And individual elements are called **atoms**. Pig also supports **maps** in the format (key#value).

These odd structures leave the programmer scratching their head wondering how to unwind all of that so they can query individual atoms. We explain some of those operations below.

Using Pig and how it works

What Pig does is run MapReduce operations across datasets. MapReduce is the fundamental concept behind Hadoop and big data in general. But it means something quite different in Hadoop than, for example, Apache Spark or the Scala programming language. In Hadoop, the **map** operation means to split datasets into pieces and work on those pieces in parallel. **Reduce** means put them all back together to deliver the desired dataset. In Spark and Scala, **map** means to run some operation on every element on a list. **Reduce** means to calculate a final single value from those operations in Spark.

You start Pig in local model using:

```
pig -x local
```

Instead of just Pig:

```
pig
```

Which causes it to run in cluster (aka mapReduce) mode. Local model simulates a distributed architecture. In cluster mode, mapReduce uses Apache Yarn to run jobs on the cluster (i.e., a network of machines) and stores the resulting data in HDFS (Hadoop Distributed File System).

Common mistakes

Pig you could say has some quirks that makes it difficult to debug errors, plus the people who maintain Pig keep changing the language, slightly. It does not usually give simple error messages, but shows a lengthy hard-to-read Java stack trace in most cases. (It will save those as a tmp* file in the current directory.) So, before we start looking at some code examples, let's look at some common mistakes that might cause you problems and frustration right up front:

- You need to check what port HDFS is running on in order to open files there. Look in $HADOOP_HOME/etc/hadoop/core-site.xml for fs.default.namehdfs://localhost:9000 to get the port number.
- Be careful copying and pasting left and quote marks from Google Docs or other word processor into the Pig shell. Pig will throw an error unless you use the proper single quote ' (ASCII 27).
- Either run Pig as root or spend time sorting through permissions issues.
- Don't use (control)(-c-) when you type an incomplete command as it prompts for more input. That will exit the shell. Just type "," or ";" until it quits the current command. Sometimes that can take quite a few tries.
- Type commands in a text editor and paste them in Pig as Pig does not let you edit previous commands easily. For example you cannot use the up cursor and then edit the previous line if

it wraps around. You can see the history of commands by typing **history**.
- If you have loaded Hadoop and Pig on your laptop, in order to learn it, run pig in local mode by typing **pig -x local** if you get any java.net.ConnectException errors. That means Yarn or some service is probably not running or configured correctly.
- Even though we called Pig easy, it can be frustrating. For example, in writing this document we found that **store** is a reserved keyword and cannot be used as a column name. And in writing the sales report we wrote below it took some time to figure that out. Pig did not give any friendly error message like "store is a reserved keyword."

Load a file

Supposed we have a data file sales.csv like this, with sales by employee by shop. (As we just said, we cannot use store for a column name as it is a reserved word. So we use **shop**.)

Shop,Employee,Sales

```
Dallas,Sam,30000
Dallas,Fred,40000
Dallas,Jane,20000
Houston,Jim,75000
Houston,Bob,65000
New York,Earl,40000
```

We can use Pig to calculate sales by store. First delete the first line that contains the header. Otherwise it will become a row in our data set. Then copy the file to the Hadoop Distributed File system like this:

```
hdfs dfs -put sales.csv  /user/hadoop/
```

Now we load it to dataset a. In Pig, datasets are called **relations**. Here we tell it that the input file is comma-delimited and we assign a schema to the data set using AS. We also give the field types: chararray and int.

```
a = LOAD 'hdfs://localhost:9000/user/hadoop/sales.csv' USING PigStorage(',')
AS (shop:chararray,employee:chararray,sales:int);
```

Then we have this tuple which we can see by typing:

```
dump a

(Dallas,Sam,30000)
(Dallas,Fred,40000)
(Dallas,Jane,20000)
(Houston,Jim,75000)
(Houston,Bob,65000)
(New York,Earl,40000)
```

Note that we could have left off the schema in the relation. In that case you would refer to fields by their relative position $0, $1, $2, … Note also that Pig is said to be lazy. That means it does not actually run the mapReduce function until it is needed, such as printing out the data with dump.

We ask Pig to show us the schema of the data set using describe:

```
describe a
```

```
a: {shop: chararray,employee: chararray,sales: int}
```

Now we group the data by shop.

```
b = GROUP a BY shop;
```

Then we run a group operation, like count, sum, or average:

```
c = FOREACH b GENERATE group as shop, SUM (a.sales);
```

This gives us sales total by each shop:

```
(Dallas,90000)
(Houston,140000)
(New York,40000)
```

Notice that we referred to the sales column as a.sales even after we created the b relation.

ForEach

Pig lets you define user defined functions (UDF) to run over elements in a tuple in the FOREACH construct. You can code those in Pig, Python, Ruby, or other. And you can use so-called **Piggy Bank** functions that other persons have written and contributed here or use DataFu. Pig does not have a lot of built-in functions, which is why people are adding their own.

But it does simple arithmetic. For example, here we multiply each sales figure by 2 creating the new dataset b. Notice that we also dropped one column and only took two fields from the original dataset a to create the dataset b.

```
b = FOREACH a GENERATE shop,sales*2
```

Flatten

Suppose we load another dataset:

```
a = LOAD 'hdfs://localhost:9000/user/hadoop/election/health.csv'  USING
PigStorage(',') As (procedure, provider, name, address, city, state, zipcode,
stateName, discharges, aveCharges, avePayments, aveMedicarePayments);
```

That took that comma-delimited file and made this simple tuple structure.

```
describe a
a: {procedure: bytearray,provider: bytearray,name: bytearray,address:
bytearray,city: bytearray,state: bytearray,zipcode: bytearray,stateName:
```

```
bytearray,discharges:
bytearray,aveCharges: bytearray,avePayments: bytearray,aveMedicarePayments:
bytearray}
```

Notice also that it made each field of type **bytearray**, which is the default if you do not define that explicitly.

Now we run a function. We run the STRSPLIT function which generates two values. Here we tell it to split the field **aveCharges** field by the "$" sign (hex 24). That field is a string for currency $99999. We cannot do math with that. So we split off the $ sign. STRSPLIT returns a tuple with a field on the left and one on the right. In this case the field on the left is empty since the $ sign is the first character. And the field on the right is the numeric part of the string.

```
b = FOREACH a GENERATE (procedure),  STRSPLIT(aveCharges, '\\u0024');
```

Now we have a tuple of a tuples, which is called a **bag**, plus the field **procedure**.

```
dump b
((948 - SIGNS & SYMPTOMS W/O MCC,$34774.21),(,34774.21))
```

So we flatten the second tuple:

```
c = FOREACH b GENERATE procedure, FLATTEN($1);
```

Generating this tuple of elements, to make it easier to work with.

```
(948 - SIGNS & SYMPTOMS W/O MCC,,15042.00)
```

Filter

Going back to our original dataset:

```
(Dallas,Sam,30000)
(Dallas,Fred,40000)
(Dallas,Jane,20000)
(Houston,Jim,75000)
(Houston,Bob,65000)
New York,Earl,40000)
```

We select only employees who have sold more that $40,000.

```
x = filter a by sales > 40000
```

Yields:

```
(Houston,Jim,75000)
(Houston,Bob,65000)
```

Join and other set operations

There is no intersection operation in Pig, but there is join. Also there is union, cogroup, cross, union, and a few others. Here, for example, is how join works. Suppose we have a list of students and what

class they are taking and then a list of classes and what each costs.

```
students = LOAD '/root/Downloads/people.txt' USING PigStorage(',') AS
(name,class);

(John,German)
(Jane,German)
(Bob,French)

class = LOAD '/root/Downloads/class.txt' USING PigStorage(',') AS
(class,fee);

(German,$100)
(French,$200)
```

Then join them by some element that is common to both tuples.

```
tuition = join students by class, class by class;

(Bob,French,French,$200)
(Jane,German,German,$100)
(John,German,German,$100)
```

That shows the class and the cost all in one structure.

Data types

Pig has a small set of data types. For example, it does not have a date data type. Instead we have float, boolean, double, bag, map, string, int, and a few more. Often it is necessary to cast a data type, like this: (int) x, or define its type, like this: x:int.

Save relations

If you shut down the Pig shell (You use **quit** or to do that.) you loose your datasets. But it saves the history so that you can recreate your work. To save your results permanently use:

```
store x into 'someName
```

The log shown on the screen will say something lie:

```
Successfully stored 2 records in: "file:///root/Downloads/someName"
```

Which you can view like this:

```
cat output

Houston    Jim    75000
Houston    Bob    65000
```

Which when we load back into Pig is a tuple again:

```
(Houston,Jim,75000)
(Houston,Bob,65000)
```

So Pig is a great tool for doing ETL (extract, transfer, load) operations on Hadoop data. Pig programs are much shorter than Java code. And its basic principles are not complicated to learn. Simplicity, in fact, is what Yahoo was looking for when they created the language for their own in-house use.