Lecture 5 - Apache Pig

BDAT 1002

Review

- Two important concepts in Hadoop
  - HDFS
  - MapReduce
- Apache Pig

Apache Pig Introduction

- Takes a set of instructions from the user
- Converts these instructions to MapReduce job
- Executes the MapReduce job in the cluster
- In this lesson we want to know two things:
  - What is Apache Pig?
  - How does it help you in the world of Hadoop?

Problem With MapReduce

- Start off by exploring what is wrong with writing a MapReduce program?
- Why do we need a tool like Apache Pig to translate our instructions to MapReduce?

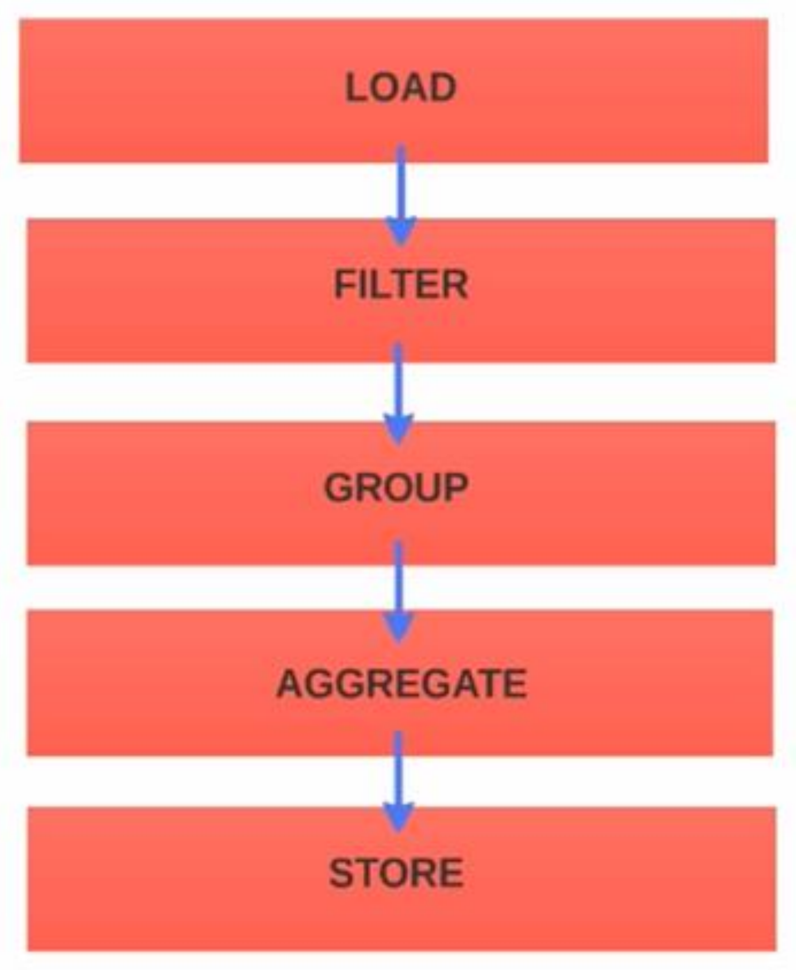
Problem With MapReduce

- There are some challenges with MapReduce programming

1) The ability to conceptually visualize the problem in MapReduce → not natural
2) Knowledge of a programming language like Java, Python, C++ etc
3) Programming requires a lot of time and effort to do simple tasks → ex joins
4) Time and effort

What's the catch?

- How can a tool replace programming and a programmer?
  - Most data analysis problems can be broken down to list of operations
  - Pig provides instructions for each operation

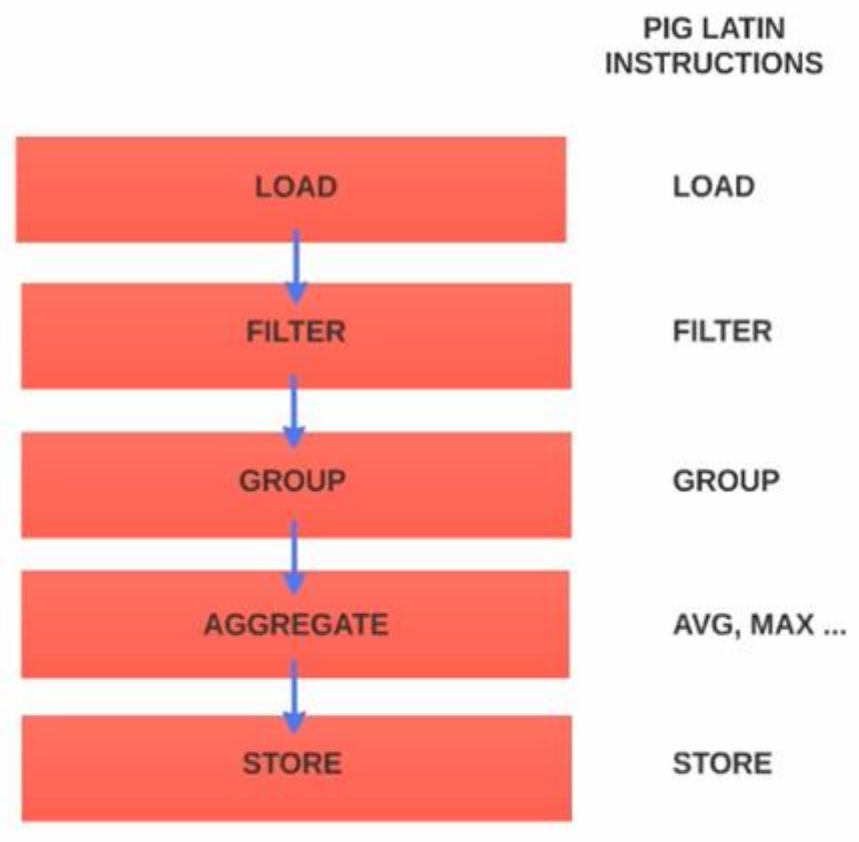


Apache Pig History

- Developed at Yahoo!
- First release 2008
- Name?
  - Not an acronym
  - Short and sweat
- Pig is a client tool
  - You don't have to install Pig on all your nodes
- Uses MapReduce and HDFS
  - Same phases

Pig Latin

- Simple to use data flow language
- As a user, you will write a series of instructions using Pig Latin
- When you **execute** the Pig instructions, Pig will analyze and optimize the instruction and then
- Translate instructions to MR jobs



Apache Pig Philosophies

- Developers at Yahoo felt that Pig needs to adhere to 4 philosophies
- The philosophies give good insight on what the tool can do
  - Even though they sound a bit funny

Philosophy # 1

- Pigs eat anything
- Pig can work with data even when you don't specify the structure of the data
  - Metadata, schema → table headings and types
- With limited instructions, Pig can understand and process the data
- Pig work very well with unstructured data
- Pig is very forgiving when not all your data in your dataset adheres to a strict schema

Philosophy # 2

- Pigs fly
- Pig was built from ground up with Big Data performance requirements in mind
- Pig has an optimizer that can rearrange operators to optimize performance
- New requirements and enhancements are made with performance requirements in mind

Philosophy # 3

- Pigs are domestic animals
- Pig is highly configurable
- Pig allows you to write user defined functions in Java and easily integrate the code
- So you are not stuck with functions and operators supplied by Pig

Philosophy # 4

- Pigs live anywhere
- Pig was envisioned to be a language for parallel data processing
- It is not tied to one particular framework like Hadoop
  - So far though just Hadoop!

Pig

- If you are hoping for a career in Hadoop, Pig is a must know tool
- It is simple and easy to learn

Summary

- Apache Pig allows MapReduce jobs to be created with ease in Hadoop

Basic Pig Instructions

Pig Latin Instructions

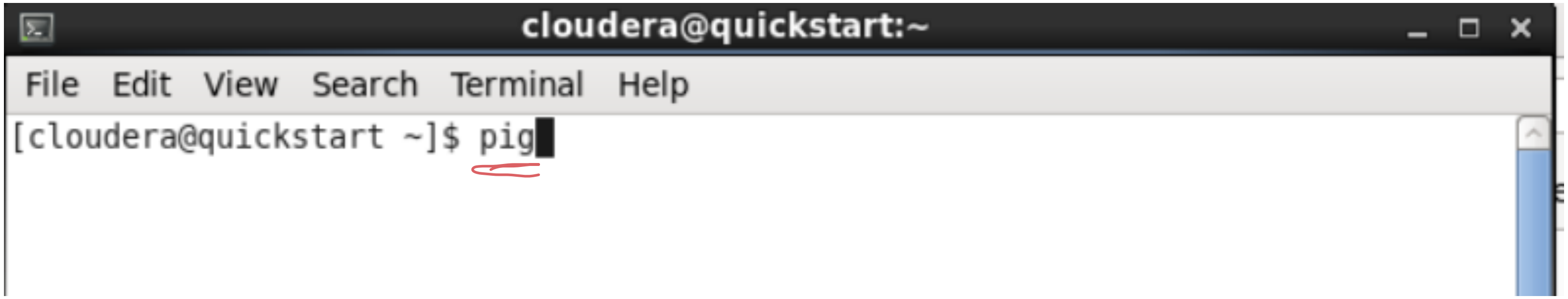
- Load datasets
- Project and manipulate columns
- Print and store the result set
- Data type conversions
- We will be using the stock dataset
  - But first let's familiarize ourselves with it

Reviewing the stock dataset

- Each value in the file is delimited by a comma
- The stock dataset is a comma delimited textfile with information about stocks traded on an exchange for each day

Grunt shell

- To interactively work with Pig, you have to enter the ***grunt shell***
- Simply type in pig in the terminal to access the grunt shell



```
[cloudera@quickstart ~]$ pig
```

- In the grunt shell, we can try out Pig Latin instructions

Pig grunt

- Let's try to load the stocks dataset and see how we can project and manipulate columns from the dataset using Pig Latin
- To work with the dataset in Pig, we first need to load the dataset
- Here is the command

Variable '\t' default value

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
      PigStorage(',') AS (exchange:chararray,
      symbol:chararray, date:datetime, open:float,
      high:float, low:float, close:float, volume:int,
      adj_close:float);
```

Pig Latin Language Introduction

- Each Pig instruction will transform the dataset one way or another
  - So how do you refer to the transformed dataset?
  - Simple you assign a name to it
- In Pig Latin a dataset is referred to as a ***relation***

Pig Latin Language Introduction

- To know about the structure of the relation, you can use the describe instruction

```
grunt> DESCRIBE stocks;
```

Pig Latin Language Introduction

- Now let's say I want to project or derive three columns from the stocks relations
- These columns are:
  - ***The symbol column***
  - ***A new column that consists of a few letters form the exchange column***
  - ***The difference between the open and close price as a new column***
- Whenever we like to project some columns from a relation, we use the FOREACH operator

Pig Latin Language Introduction

- Here is the instruction

```
grunt> projection = FOREACH stocks GENERATE
      symbol, SUBSTRING($0, 0, 1) AS sub_exch,
      close - open AS up_or_down;
```

subtraction.

Pig Latin Language Introduction

- Notice that so far, there was no MapReduce job
- Let's say we want to print this result set, here is the instruction

```
grunt> DUMP projection;
```

- This instruction will result in a MapReduce job and will print the result set to the screen
- The operator we used was DUMP

Pig Latin Language Introduction

- It is not always ideal to print the result to the screen
- Most of the time what we need is to store the result in HDFS
- In this case, use the STORE operator

```
grunt> STORE projection INTO /BDAT1002/projection
```

- This will run a MapReduce job as well, can track the operation in the web UI (port 19888)

Different Load Operations

- Let's look as some other load ways to project our data → all equivalent

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',');
```

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',’) AS (exchange, symbol, date, open,
high, low, close, volume, adj_close);
```

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',’) AS (exchange:chararray,
symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float,
volume:int, adj_close:float);
```

Load Operations

- Which one do you prefer?
  - Third one is preferred
  - Improves readability and usability of your pig Script
- Always define your dataset properly with column names and data types

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',’) AS (exchange:chararray,
symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float,
volume:int, adj_close:float);
```

Load Operations

- But aren't you curious to know how columns are projected with first load instruction?

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',');
```

- Let's run the first load instruction and do a describe on the relation

```
grunt> DESCRIBE stocks;
```

  - "Schema not known"

Load Operations

- So Pig does not know the schema because the developer did not provide one
- So question is what if you want to project columns from a dataset that the schema is not known
  - You can project using the position of the column in the dataset

```
grunt> projection = FOREACH stocks GENERATE $1 AS
symbol, SUBSTRING($0, 0, 1) AS sub_exch, $6 - $3 AS
up_or_down;
```

↳ column's location.

no data type.

Load Operations

- Use describe command and see the output
  - What type of datatypes are there?
  - Where does this come from?
- But how does Pig know the data type?

Load Operations

- When there is no datatype specified, Pig uses implicit casting
  - By default Pig chooses data type as bytearray but then converts it to other types depending on operation occurring in your instruction
  - For example, columns 7 and 4 are converted to double since there is a subtraction
  - Column 2 is a charray because we use function SUBSTRING
- Why double and not int?
  - Since you might lose precision

This means if you have long number, you will be able to store partial values.

Different Load Operations

- Now try the second load instruction

```
grunt> stocks = LOAD '/BDAT1002/stocks' USING
PigStorage(',') AS (exchange, symbol, date, open,
high, low, close, volume, adj_close);
```

- And use the describe command to see what kind of datatypes are assumed
  - All the data types will be bytearray

by default

Different Load Operations

- Pig is also forgiving when the datatype does not match the datatype you have mentioned
- In this case it will put NULL
- Generally though, although Pig does a good job in assigning and converting data types, it is not a good idea to leave things to Pig
  - Can lead to unexpected errors when conversions don't match
- ***Always assign column names and proper data types***

Summary

- We looked at
  - loading and projecting datasets
  - how Pig handles data sets when we don't specify the column names or data types
  - the default data type assignment → bytearray
  - How Pig does implicit casting when data types are not specified

Solving a Problem with Pig

Introduction

- We want to solve a real world problem using Apache Pig
- We will use a similar problem as the stock – max price problem
- We want to list the top 10 stock symbols for the year 2003 with the highest average volume by using Pig
- We will in turn learn some very useful instructions in Pig Latin →

Introduction

- We will in turn learn some very useful instructions in Pig Latin
  - Filter the data set
  - Group the data set
  - Aggregation
  - Limit the number of records
  - Ordering the data set

Solving the problem

- First load the data set

```
grunt> stocks = LOAD '/BDAT1002/pig/stocks' USING
PigStorage(',’) AS (exchange:chararray,
symbol:chararray, date:datetime, open:float,
high:float, low:float, close:float,
volume:int, adj_close:float);
```

Solving the problem

- Next, we want to filter the dataset for results in 2003 only
- We use the FILTER operator

```
grunt> filter_by_yr = FILTER stocks BY GetYear(date)
== 2003;
```

what condition

Solving the problem

- Now we want to group

```
grunt> grp_by_sym = GROUP filter_by_yr BY symbol;
```

나머지 컬럼도 같이 저장이 되나?

Solving the problem

- Now we are ready to do aggregation
- But to do aggregation, you must understand the structure of the grp_by_sym
- So use the describe operator to understand the structure

```
grunt> DESCRIBE grp_by_sym;
```

Structure of grp_by_sym

- grp_by_sym has two columns.
  - First column is called "group" with chararray data type by symbol.
  - Filter_by_year is the second column which in turn has a *collection* of records for that particular symbol 2003 filtered

```
grunt> DUMP grp_by_sym;
```

Structure of grp_by_sym

- grp_by_sym has two columns.
  - First column is called “group“ with chararray data type
  - Filter_by_year is the second column which in turn has a *collection* of records for that particular symbol

```
grunt> avg_volume = FOREACH grp_by_sym GENERATE
group, ROUND(AVG(filter_by_yr.volume)) AS
avgvolume;
```

Structure of grp_by_sym

- Next user the ORDER operator to sort the records by volume

```
grunt> avg_vol_ordered = ORDER avg_volume BY
avgvolume DESC;
```

Structure of grp_by_sym

- Once ordered, limit the records to top 10
- And store values in HDFS

```
grunt> top10 = LIMIT avg_vol_ordered 10;
grunt> STORE top10 INTO '/BDAT1002/avg-volume'
USING PigStorage(',');
```

Structure of grp_by_sym

- And store values in HDFS
  - Note that results are tab limited by default, if you want comma delimited format don't forget to use the PigStorage function

```
grunt> STORE top10 INTO '/BDAT1002/pig/avg-
volume' USING PigStorage(',');
```

Pig Scripts

- Although we can execute the instructions one by one in the grunt shell
- Grunt shell is meant for development but not for production execution
- So we will also execute all the instructions as a script outside of the grunt shell

How to execute a script

- Copy all the instructions into a file
- Save the file
  - you do not need to have .pig extension but helpful
- To execute the script, simply use "pig" and the location and name of the file

```
[cloudera@quickstart ~]$> pig average-volume.pig
```

How to execute a script

- Once you execute the script several things happen behind the scene
  - Pig analyzes all the instructions
  - Applies optimizations if possible
  - Prepares an execution plan
  - Translates the instructions into one or more MapReduce jobs

Look at the results

- Look at the results

```
[cloudera@quickstart ~]$> hadoop fs -cat
/BDAT1002/pig/avg-volume/part-r-00000
```

- Can also look at the web UI at localhost:19888

Production Environment Details

- In our script, the file locations are “hard coded”
- What if you want to change the locations?
  - You have to change the script
- If the script is running in a production environment, this is not efficient
- We can avoid this by using parameters

Production Environment Details

- As part of the development environment, you should always test your script
  - You want to check to see if there are errors
- Instead of running an untested pig script in the Hadoop cluster, you can first run it locally
- To do this, use the flag –x local
  - The input and output locations you specify are not in HDFS anymore
  - Also try to run the script on a smaller representative dataset

Summary

- In this section we learned:
  - Filtering, Grouping, Ordering operations in Pig Latin
  - How to run a Pig script
  - How to use parameters in scripts