Pig

Definitions

- Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs.

Why Pig?

- An abstraction on top of Hadoop Map Reduce.
- MapReduce is a powerful programming model for parallelism based on rigid procedural structure.
- Hadoop MapReduce allows programmers to filter and aggregate data from HDFS to gain meaningful insights from big data.
- The Map and Reduce algorithmic functions can be implemented using C, Python and Java.
- The only drawback to use the coding approach of Hadoop MapReduce is that hadoop developers need to write several lines of basic java code requiring extra effort and time for code review and QA. Thus, to simplify this frameworks such has Pig have evolved shorten the times for development and testing an analytic job.

Why Pig?

- By using Hadoop MapReduce as the coding approach - it is hard to achieve join functionality making it difficult and time consuming to implement complex business logic.
- There is lot of development effort required to decide on how different Map and Reduce joins will take place and there could be chances that hadoop developers might not be able to map the data into the particular schema format.
- However, the advantage is that MapReduce provides more control for writing complex business logic when compared to Pig.

Comparision – WordCount - MR

```
package com.ixat.mr.WordCount;

import java.io.IOException;
….

public class WordCount
{
    public static void main( String[]
args ) throws IllegalArgumentException,
IOException, ClassNotFoundException,
InterruptedException
    {
            int numReducers = 2;
         Configuration conf = new
Configuration();
          String[] otherArgs = new
GenericOptionsParser(conf,
args).getRemainingArgs();
          if (otherArgs.length < 2) {

System.err.println("Usage: WordCount
<in> <out> [numReducers]");
              System.exit(2);
          }

          if(otherArgs.length>2){
               numReducers =
Integer.parseInt(otherArgs[2]);
          }
          Job job =
Job.getInstance(conf);
          job.setJobName("WordCount");


job.setJarByClass(WordCount.class);

job.setMapperClass(TokenizerMapper.clas
s);

job.setReducerClass(IntSumReducer.class
);



job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.cla
ss);


job.setNumReduceTasks(numReducers);

FileInputFormat.addInputPath(job, new
Path(otherArgs[0]));

FileOutputFormat.setOutputPath(job, new
Path(otherArgs[1]));


System.exit(job.waitForCompletion(true)
? 0 : 1);
          }

    public static class TokenizerMapper
extends Mapper<Object, Text, Text,
IntWritable> {

        private final static
IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(Object key,
Text value, Context context) throws
IOException, InterruptedException {

            String cleanLine =
value.toString().toLowerCase().replaceA
ll("[_|$#<>\\^=\\[\\]\\*/\\\\,;,.\\-
:()?!\"']", " ");

            StringTokenizer itr = new
StringTokenizer(cleanLine);
            while (itr.hasMoreTokens())
{

word.set(itr.nextToken().trim());
                context.write(word,
one);
            }

        }
    }


    public static class IntSumReducer
extends Reducer<Text, IntWritable,
Text, IntWritable> {
        private IntWritable result =
new IntWritable();

        @Override
        public void reduce(Text key,
Iterable<IntWritable> values, Context
context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val :
values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
```

Comparision – WordCount - Pig

```
input_lines = LOAD '/test/pg....txt' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS
count, group AS word;
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/OUTPUT/results.txt';
```

Installation

- Download Pig from http://www.eu.apache.org/dist/pig/pig-0.14.0/pig-0.14.0.tar.gz

Check the release note in the tar.gz file for version compatability

Copy the .gz to your CentOS (or) do a wget http://www.eu.apache.org/dist/pig/pig-0.14.0/pig-0.14.0.tar.gz

- unzip

  tar xzf pig-0.14.0.tar.gz

- rename the folder

  mv pig-0.14.0 pig

If JAVA_HOME is not set, set it now

set HADOOP_HOME to appropriate hadoop dir.

set PIG_HOME and point this to the directory where you have unzipped pig

Example .bashrc

export JAVA_HOME=/opt/jdk1.8.0_66

export JRE_HOME=/opt/jdk1.8.0_66/jre

export PATH=$PATH:/opt/jdk1.8.0_66/bin:/opt/jdk1.8.0_66/jre/bin

export HADOOP_HOME=/home/hdtester/hadoop

….

For PIG assuming you have unzipped Pig to /home/hdtester/pig, set the below in your bashrc and do a source on bashrc (or relogin)

export PIG_HOME=/home/hdtester/pig

export PATH=$PATH:$PIG_HOME/bin

Running Pig

- **Grunt Shell:** Enter Pig commands manually using Pig's interactive shell, Grunt.
- **Script File:** Place Pig commands in a script file and run the script.
- **Embedded Program:** Embed Pig commands in a host language and run the program

Run Modes

- Local Mode: Would use local filesystem, for debugging and easy start (-x local)
- Hadoop (mapreduce) Mode: To run Pig in hadoop (mapreduce) mode, you need access to a Hadoop cluster and HDFS installation, the default mode (or) use –x mapreduce
- Also, we have tez –x tez.

Sample PIG script

Start pig in local mode (pig -x local), this would open grunt prompt.

Grunt prompt is an interactive shell for Pig Commands.

```
A = load '/etc/passwd' using PigStorage(':');
B = foreach A generate $0 as ID, $5 as HOME;
dump B;
```

One more example

- Pig has some tutorials and sample data
- One such sample file is $PIG_HOME/tutorial/data/excite-small.log
- The file has some search queries done by users on Excite search engine
- The structure of this file is UserID, TimeStamp and Search Query.
- Example data, first 3 lines

```
head -3 $PIG_HOME/tutorial/data/excite-small.log
        2A9EABFB35F5B954        970916105432    +md foods +proteins
        BED75271605EBD0C        970916001949    yahoo chat
        BED75271605EBD0C        970916001954    yahoo chat
```

Order searches done on excite

- Start Pig in Local Mode, also supply a parameter named P_H which would carry the Pig Install directory name

```
pig -x local -param P_H=$PIG_HOME
```

- In the Grunt prompt do the below (more on the syntax later)

```
A = LOAD '$P_H/tutorial/data/excite-small.log' USING PigStorage('\t') AS (user, time, query);
describe A;
B = Group A by query;
describe B;
C = FOREACH B GENERATE group, COUNT(A.query);
D = ORDER C BY $1 DESC;
describe D;
store D into './OUTPUT' ;
explain D;
Illustrate D;
```

- Exit grunt prompt using CTRL+D, examine the contents of OUTPUT dir

Lets do the same thing in MR mode

- Start HDFS, Yarn and HistoryServer.
- Copy `excite-small.log` to HDFS under /pigtest dir of HDFS
- Start PIG

```
pig -param input_path=/pigtest/excite-small.log
```

- In the Grunt prompt do the below (more on the syntax later)

```
A = LOAD '$input_path' USING PigStorage('\t') AS (user, time, query);
B = Group A by query;
C = FOREACH B GENERATE group, COUNT(A.query);
D = ORDER C BY $1 DESC;
store D into '/pigtest/OUTPUT' ;
```

- Exit grunt prompt using CTRL+D and examine what you have in /pigtest/OUTPUT folder in HDFS

Pig Types

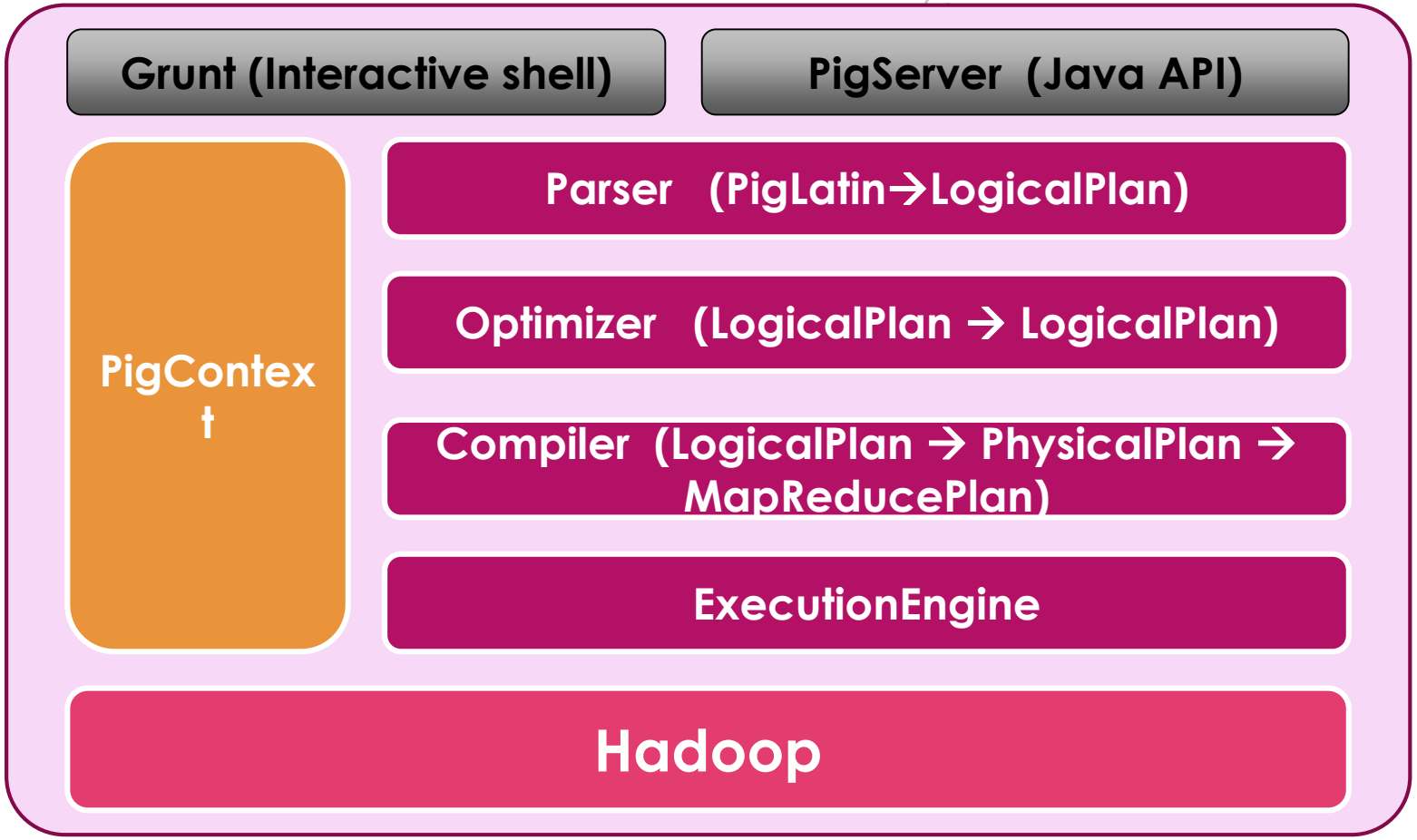
Pig Type	Java Class
bytearray	DataByteArray
chararray	String
int	Integer
long	Long
float	Float
double	Double
tuple	Tuple
bag	DataBag
map	Map<Object, Object>

Reference

- https://github.com/iXat-Training/Hadoop101/blob/master/13_Pig/Pig-Reference.pdf

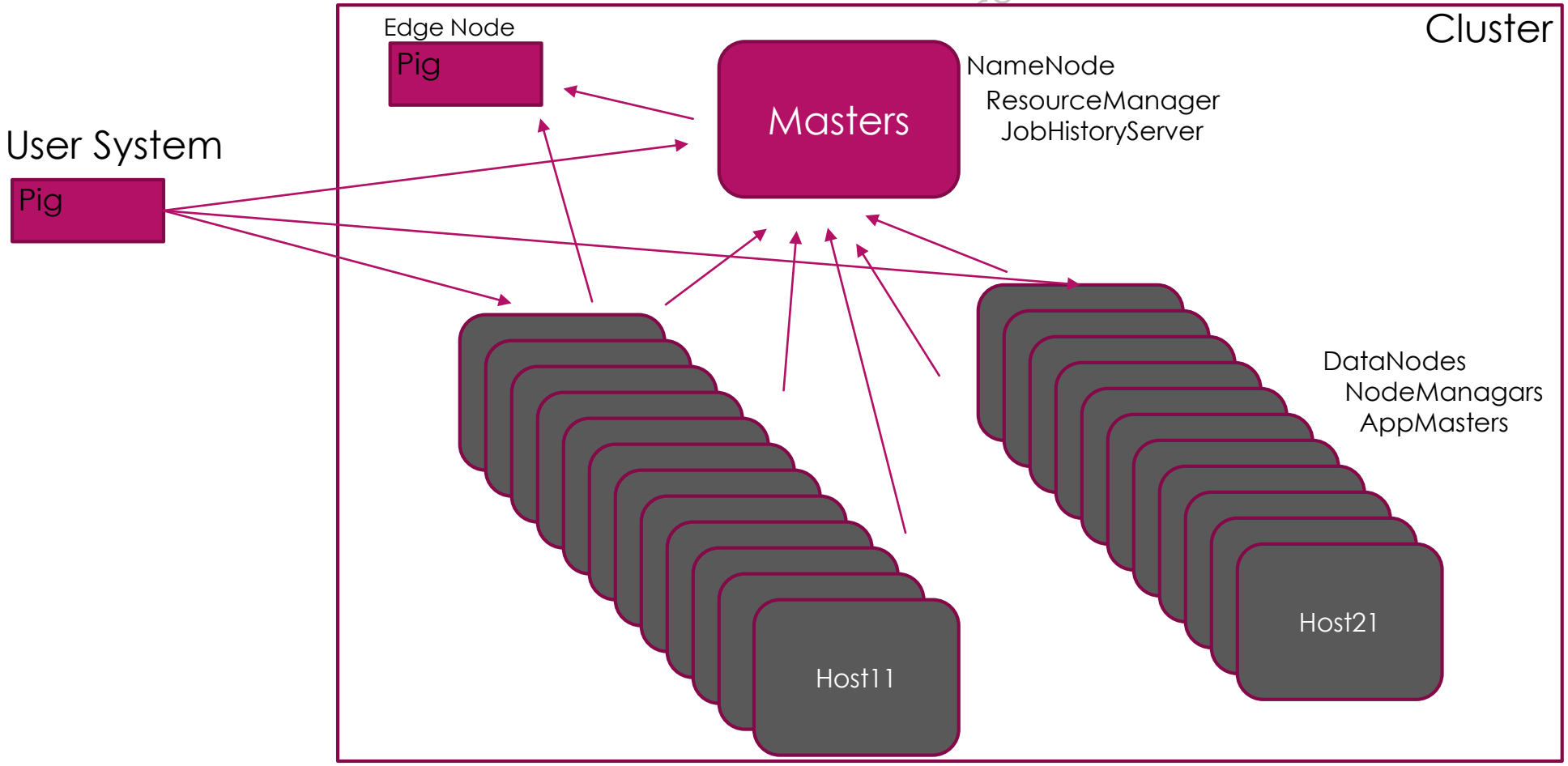
- http://pig.apache.org/docs/r0.14.0/basic.html

Architecture (MR mode)



Architecture (MR mode)

Pig Physical Architecture



Some more examples

- AdventureWorks
  - Sample Data set http://download-codeplex.sec.s-msft.com/Download/Release?ProjectName=msftdbprodsamples&DownloadId=880662&FileTime=130536511930200000&Build=21031
  - Schema of Employees Data (data in Employee*.csv)

Employee	EmpSal	EmpDept
EmpID	EmpID	BusinessEntityID
NationalIDNumber	RateChangeDate	DepartmentID
LoginID	Rate	ShiftID
OrganizationNode	PayFrequency	StartDate
OrganizationLevel	ModifiedDate	EndDate
JobTitle		ModifiedDate
BirthDate		
MaritalStatus		
Gender		
HireDate		
SalariedFlag		
VacationHours		
SickLeaveHours		
CurrentFlag		
rowguid		
ModifiedDate		

Loading the data with schema

- emp = load 'employee.tsv' USING PigStorage('\t') as (empid:int, nationalidnumber:chararray,loginid:chararray,organizationnode:chararray ,organizationlevel:int,jobtitle:chararray,birthdate:chararray,maritalstatus:ch ararray,gender:chararray,hiredate:chararray,salariedflag:int,vacationhour s:int,sickleavehours:int,currentflag:int,rowguid:chararray,modifieddate:cha rarray);
- sal = load 'empsal.tsv' USING PigStorage('\t') as (empid:int,ratechangedate:chararray,rate:double,payfrequency:int,modif ieddate:chararray);
- dept = load 'empdept.tsv' USING PigStorage('\t') as (empid:int,departmentid:int,shiftid:int,startdate:chararray,enddate:chararr ay,modifieddate:chararray);

Examples - Operators

- Find super boss

```
superboss = filter emp by organizationnode is null;
```

- Dump all female workers

```
females = filter emp by UPPER(gender)=='F';
dump females;
```

- Find count of female workers

```
females = filter emp by UPPER(gender)=='F';
fg = group females by gender;
cfg = foreach fg generate COUNT(females);
```

- count of female workers by marital status

```
females = filter emp by UPPER(gender)=='F';
fg = group females by (maritalstatus,gender);
cfgm = foreach fg generate group, COUNT(females.maritalstatus);
```

Examples Join

- Join employees and salary data sets

```
empsal = join emp by empid, sal by empid;

empsal_f = foreach empsal generate emp::empid, emp::loginid, sal::rate;

dump empsal_f;
```

- Now find all employees who have taken increments more than once.

```
empsal_g = group empsal_f by emp::empid;

empsal_multi = foreach empsal_g generate group, COUNT(empsal_f) as
numicr;

empsal_multio = order empsal_multi by numicr desc;

empsal_multiof = filter empsal_multio by numicr >1;
```

Extensions using PiggyBank

- What if you have data in XML format?
  - Extend the capabilities of Pig using UDF's (User Defined Functions)
  - Some UDF's that are already implemented are termed Piggybank.
  - You could also create your own UDF.

  - Load UDF into Pig using register keyword.
  - If in MR Mode, ensure the jar is in classpath
    - One way is to upload the UDF Jar to HDFS and Register it via HDFS path

XML Load via PiggyBank

Sample to load employee XML, pick the XML from our Github
The sample demonstrates in local mode, run the script by setting pdir parm to PIG_HOME

```
register '$pdir/contrib/piggybank/java/piggybank.jar'

DEFINE XPath org.apache.pig.piggybank.evaluation.xml.XPath();

empxml =  load 'emp.xml' using org.apache.pig.piggybank.storage.XMLLoader('employee') as
(thexml:chararray);

empdata = FOREACH empxml GENERATE XPath(thexml, 'employee/Name'), XPath(thexml,
'employee/Job'), XPath(thexml, 'employee/Email'), XPath(thexml, 'employee/Salary'),
XPath(thexml, 'employee/Gender'), XPath(thexml,'employee/MaritalStatus');

dump empdata;
```

Assignment – use a JSON Loader

- Clues –

```
Register PiggyBank

Use the jsonloader from org.apache.pig.builtin.JsonLoader();

Define the JsonLoader using

    define JsonLoader org.apache.pig.builtin.JsonLoader();

You can pass the schema direct to JsonLoader, for example

    Load 'some.json' using  JSONLoader('fieldName:datatype,
fildName:dataType....')
```