**Apache PIG - Lab practiced**

**students.txt**

John 21 3.89 ewrew  
Sally 19 2.56 qwewr  
Alice 22 3.76 rewrwer  
Doug 19 1.98 vxcbn  
Susan 26 3.25 awe334

grunt> A = LOAD 'students.txt' AS (name:chararray, age:int, gpa:float);

grunt> describe A;

A: {name: chararray,age: int,gpa: float}

grunt> R = FILTER A by (age>=20) and (gpa>=3.5);

--> PIG is running MapReduce Jobs behind. All of the commands are converted into MapReduce Jobs.

grunt> illustrate; 0R grunt> illustrate R;

grunt> R = FOREACH A GENERATE age,gpa;

grunt> B = GROUP A by age;

grunt> dump B;

(19,{(Doug,19,1.98),(Sally,19,2.56)})

(21,{(John,21,3.89)})

(22,{(Alice,22,3.76)})

(26,{(Susan,26,3.25)})

grunt> describe B;

B: {group: int,A: {(name: chararray,age: int,gpa: float)}}

grunt> C = FOREACH B GENERATE group, A.name;

grunt> dump C;

(19,{(Doug),(Sally)})

(21,{(John)})

(22,{(Alice)})

(26,{(Susan)})

grunt> STORE B into 'outputdir2' USING PigStorage('|');

[khasimbabueai2500@ip-172-31-20-58 PIGFiles]$ pig -x local students.pig

**NYSE Dividends file (NYSE\_dividends.txt)**

**average\_dividend.pig**

stockfile\_load = load '/home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange:chararray,stock:chararray,date:int,dividend:double);

grp\_by\_stock = group stockfile\_load by stock;

dividend\_avg = foreach grp\_by\_stock generate group,AVG(stockfile\_load.dividend);

dividend\_avg\_limit = LIMIT dividend\_avg 10;

store dividend\_avg\_limit into '/home/cloudera/khasimbabu/PIG\_Excercise/average\_dividend' USING PigStorage('|');

[cloudera@quickstart PIG\_Excercise]$ pig -x local average\_dividend.pig

[cloudera@quickstart PIG\_Excercise]$ cat average\_dividend/part-r-00000 | head -5

CA|0.04

CB|0.35

CE|0.04

CF|0.1

CI|0.04

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The only thing Pig needs to know to run on your cluster is the location of your cluster’s.

NameNode and JobTracker. The NameNode is the manager of HDFS, and the JobTracker coordinates MapReduce jobs. In Hadoop 0.18 and earlier, these locations are found in your hadoop-site.xml file. In Hadoop 0.20 and later, they are in three separate files: core-site.xml, hdfs-site.xml, and mapred-site.xml.

"your gateway machine" = The machine from which you are launching Pig jobs.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**--no\_schema.pig**

daily = load '/home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt';

grunt> calcs = foreach daily generate $7 / 1000, $3 \* 100.0, SUBSTRING($0, 0, 1), $6 - $3;

grunt> fltrd = filter daily by $6 > $3;

grunt> A = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange, symbol, date, dividend);

grunt> describe A;

A: {exchange: bytearray,symbol: bytearray,data: bytearray,dividend: bytearray}

grunt> B = filter A by dividend > 0.6; (or) grunt> B = filter A by $3 > 0.6;

grunt> C = foreach B generate LOWER(symbol);

grunt> divs = load 'NYSE\_dividends' using PigStorage(',');

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

prices = load 'NYSE\_daily' as (exchange, symbol, date, open, high, low, close, volume, adj\_close);

beginning = foreach prices generate ..open; -- produces exchange, symbol, date, open

middle = foreach prices generate open..close; -- produces open, high, low, close

end = foreach prices generate volume..; -- produces volume, adj\_close

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

grunt> stocks = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange:chararray, symbol:chararray,

date:chararray, dividends:float);

grunt> startswithcm = filter stocks by symbol matches 'CM.\*';

grunt> notstartswithcm = filter stocks by not symbol matches 'CM.\*';

grunt> stock\_data = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange,stock);

grunt> describe stock\_data;

stock\_data: {exchange: bytearray,stock: bytearray}

grunt> stock\_data\_grp = group stock\_data by stock;

grunt> describe stock\_data\_grp;

stock\_data\_grp: {group: bytearray,stock\_data: {(exchange: bytearray,stock: bytearray)}}

grunt> stock\_data\_grp\_count = foreach stock\_data\_grp generate group, COUNT(stock\_data.stock);

grunt> stock\_data\_grp\_count = foreach stock\_data\_grp generate group, COUNT(stock\_data);

**--twokey.pig**

daily = load 'NYSE\_daily' as (exchange, stock, date, dividends);

grpd = group daily by (exchange, stock);

avg = foreach grpd generate group, AVG(daily.dividends);  
  
**--countall.pig**

grunt> stock\_data = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange,stock);

grunt> P = group stock\_data all;

grunt> Q = foreach P generate COUNT(stock\_data);  
  
**--order.pig**

grunt> A = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange, symbol, date, dividend);

grunt> B = ORDER A by date;

grunt> C = Order A by date, symbol;

grunt> D = Order A by date DESC, symbol;  
  
**--distinct.pig**

grunt> A = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange, symbol);

grunt> B = DISTINCT A;  
  
**--join.pig**

grunt> daily = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_daily.txt' as (exchange, symbol, date, open, high, low);

grunt> describe daily;

daily: {exchange: bytearray,symbol: bytearray,date: bytearray,open: bytearray,high: bytearray,low: bytearray}

grunt> divs = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange, symbol, date, dividend);

grunt> describe divs;

divs: {exchange: bytearray,symbol: bytearray,date: bytearray,dividend: bytearray}

grunt> tab\_join = join daily by symbol, divs by symbol;

grunt> tab\_join2 = join daily by (symbol,date), divs by (symbol,date);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

outer joins - records that do not have a match on the other side are included, with null values being filled in for the missing fields

Full outer join means - records from both sides are taken even when they do not have matches

Left outer join means - records from the left side will be included even when they do not have a match on the right side

Right outer joins means - records from the right side will be included even when they do not have a match on the left side

Inner Join – It returns row when there is a match in both the tables

Left Outer Join – This join returns all rows from the left table and matching rows from the right table

Right Outer Join – This join returns all rows from the right table and matching rows from the left table

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**--leftjoin.pig**

grunt> tab\_join\_leftouter = join daily by (symbol,date) left outer, divs by (symbol,date);  
  
**--parallel.pig**

daily = load 'NYSE\_daily' as (exchange, symbol, date, open, high, low, close, volume, adj\_close);

bysymbl = group daily by symbol parallel 10;

average = foreach bysymbl generate group, AVG(daily.close) as avg;

sorted = order average by avg desc parallel 2;  
  
**--defaultparallel.pig**

set default\_parallel 10;

**Nested foreach**

**--distinct\_symbols.pig**

grunt> A = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_daily.txt' as (exchange, symbol);

grunt> B = group A by exchange;

grunt> C = foreach B {

>> sym = A.symbol;

>> uniq\_sym = distinct sym;

>> generate group, COUNT(uniq\_sym);

>> };

grunt> describe C;

C: {group: bytearray,long}  
  
**--analyze\_stock.pig**

register 'acme.jar';

define analyze com.acme.financial.AnalyzeStock();

daily = load 'NYSE\_daily' as (exchange:chararray, symbol:chararray,

date:chararray, open:float, high:float, low:float,

close:float, volume:int, adj\_close:float);

grpd = group daily by symbol;

analyzed = foreach grpd {

sorted = order daily by date;

generate group, analyze(sorted);

};  
  
**--hightest\_dividend.pig**

grunt> A = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange, symbol, date, dividend);

grunt> B = group A by symbol;

grunt> C = foreach B {

>> sorted = ORDER A by dividend desc;

>> top = LIMIT sorted 3;

>> generate group,flatten(top);

>> };

grunt> describe C;

C: {group: bytearray,top::exchange: bytearray,top::symbol: bytearray,top::date: bytearray,top::dividend: bytearray}  
  
**--double\_distinct.pig**

grunt> divs = load 'file:///home/cloudera/khasimbabu/PIG\_Excercise/NYSE\_dividends.txt' as (exchange:chararray, symbol:chararray);

grunt> grpd = group divs all;

grunt> uniq = foreach grpd {

>> exchanges = divs.exchange;

>> uniq\_exchanges = distinct exchanges;

>> symbols = divs.symbol;

>> uniq\_symbols = distinct symbols;

>> generate COUNT(uniq\_exchanges), COUNT(uniq\_symbols);

>> };

grunt> describe uniq;

uniq: {long,long}  
  
**--repljoin.pig**

jnd = join daily by (exchange, symbol), divs by (exchange, symbol)

using 'replicated';

The using 'replicated' tells Pig to use the fragment-replicate algorithm to execute this join. Because no reduce phase is necessary, all of this can be done in the map task.  
  
jnd = join cinfo by city, users by city using 'skewed';

Records from users with New York as the key would be split between two separate reducers.   
  
**--cogroup = semijoin.pig**

daily = load 'NYSE\_daily' as (exchange:chararray, symbol:chararray,

date:chararray, open:float, high:float, low:float,

close:float, volume:int, adj\_close:float);

divs = load 'NYSE\_dividends' as (exchange:chararray, symbol:chararray,

date:chararray, dividends:float);

grpd = cogroup daily by (exchange, symbol), divs by (exchange, symbol);

sjnd = filter grpd by not IsEmpty(divs);

final = foreach sjnd generate flatten(daily);  
  
**--multiquery.pig**

players = load 'baseball' as (name:chararray, team:chararray,

position:bag{t:(p:chararray)}, bat:map[]);

pwithba = foreach players generate name, team, position,

bat#'batting\_average' as batavg;

byteam = group pwithba by team;

avgbyteam = foreach byteam generate group, AVG(pwithba.batavg);

store avgbyteam into 'by\_team';

flattenpos = foreach pwithba generate name, team,

flatten(position) as position, batavg;

bypos = group flattenpos by position;

avgbypos = foreach bypos generate group, AVG(flattenpos.batavg);

store avgbypos into 'by\_position';

wlogs = load 'weblogs' as (pageid, url, timestamp);

split wlogs into apr03 if timestamp < '20110404',

apr02 if timestamp < '20110403' and timestamp > '20110401',

apr01 if timestamp < '20110402' and timestamp > '20110331';

store apr03 into '20110403';

store apr02 into '20110402';

store apr01 into '20110401';

wlogs = load 'weblogs' as (pageid, url, timestamp);

apr03 = filter wlogs by timestamp < '20110404';

apr02 = filter wlogs by timestamp < '20110403' and timestamp > '20110401';

apr01 = filter wlogs by timestamp < '20110402' and timestamp > '20110331';

store apr03 into '20110403';

store apr02 into '20110402';

store apr01 into '20110401';

Hadoop uses a class called Partitioner to partition records to reducers during the shuffle phase.

Pig does not override the default partitioner, except for order and skew join. The balancing operations in these require special Partitioners.

register acme.jar; --jar containing the partitioner

users = load 'users' as (id, age, zip);

grp = group users by id partition by com.acme.userpartitioner parallel 100;

**--daily.pig**

daily = load 'NYSE\_daily' as (exchange:chararray, symbol:chararray,

date:chararray, open:float, high:float, low:float, close:float,

volume:int, adj\_close:float);

yesterday = filter daily by date == '$DATE';

grpd = group yesterday all;

minmax = foreach grpd generate MAX(yesterday.high), MIN(yesterday.low);

pig -p DATE=2009-12-17 daily.pig

pig -param\_file daily.params daily.pig

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**--describe.pig**

divs = load 'NYSE\_dividends' as (exchange:chararray, symbol:chararray,

date:chararray, dividends:float);

trimmed = foreach divs generate symbol, dividends;

grpd = group trimmed by symbol;

avgdiv = foreach grpd generate group, AVG(trimmed.dividends);

describe trimmed;

describe grpd;

describe avgdiv;

trimmed: {symbol: chararray,dividends: float}

grpd: {group: chararray,trimmed: {(symbol: chararray,dividends: float)}}

avgdiv: {group: chararray,double}

**--explain.pig**

divs = load 'NYSE\_dividends' as (exchange, symbol, date, dividends);

grpd = group divs by symbol;

avgdiv = foreach grpd generate group, AVG(divs.dividends);

store avgdiv into 'average\_dividend';

bin/pig -x local -e 'explain -script explain.pig'

**--illustrate.pig**

divs = load 'NYSE\_dividends' as (e:chararray, s:chararray, d:chararray, div:float);

recent = filter divs by d > '2009-01-01';

trimmd = foreach recent generate s, div;

grpd = group trimmd by s;

avgdiv = foreach grpd generate group, AVG(trimmd.div);

illustrate avgdiv;

**--stats.pig**

a = load '/user/pig/tests/data/singlefile/studenttab20m' as (name, age, gpa);

b = load '/user/pig/tests/data/singlefile/votertab10k'

as (name, age, registration, contributions);

c = filter a by age < '50';

d = filter b by age < '50';

e = cogroup c by (name, age), d by (name, age) parallel 20;

f = foreach e generate flatten(c), flatten(d);

g = group f by registration parallel 20;

h = foreach g generate group, SUM(f.d::contributions);

i = order h by $1, $0 parallel 20;

store i into 'student\_voter\_info';

**Analysing Movies Dataset**

(49590 observations, 5 columns)

1) Loading of the file(movies\_data dataset) in to PIG.

grunt> movies = LOAD 'movies\_data.xls' USING PigStorage(',') AS (id:int, name:chararray, year:int, rating:double, duration:int);

grunt> movies = LOAD 'file:///home/cloudera/Khasimbabu/movies\_data.xls' USING PigStorage(',') AS (id:int, name:chararray, year:int, rating:double, duration:int);

grunt> describe movies;

movies: {id: int,name: chararray,year: int,rating: double,duration: int}  
  
--> define schema for stuctured data is recommanded, otherwise it will treated the entire row as one element. It is optional to define a schema.

--> PigStorage is a most basic function used for input and output for PIG.

--> At this point of time, no data is loaded. After dump command the data will be displayed and loaded.

grunt> dump movies;  
  
2) To display Movies greater than 4 rating.

grunt> movies\_greater\_than\_four = FILTER movies BY (float)rating>4.0;  
  
3) Remove the duplicate records

grunt> movies\_nodups = DISTINCT movies;  
  
4) How many movies are there in the file. (First we have to do Group By all movies, then run the count function)

grunt> count\_all = GROUP movies\_nodups ALL;

grunt> total\_count = FOREACH count\_all GENERATE COUNT(movies.id);

grunt> dump total\_count;

grunt> describe count\_all;

count\_all: {group: chararray,movies: {(id: int,name: chararray,year: int,rating: double,duration: int)}}

grunt> illustrate count\_all;  
  
5) Remove the Not Null records

grunt> movies\_duration\_notnull = FILTER movies by $4 is not null;  
  
--> $4 is represent Fifth column(or we can specify direct schema column name).$0 represents first column. "is not null" means only display not null column data.  
  
6) Longest movie from the dataset

grunt> movies\_duration = FOREACH movies\_duration\_notnull generate name,(double)duration/60;

grunt> describe movies\_duration;

movies\_duration: {name: chararray,double}  
  
7) Name of highest rated movie in each year.

grunt> movies\_from\_longest\_to\_shortest = ORDER movies\_duration by $1 DESC;

grunt> longest\_movies = LIMIT movies\_from\_longest\_to\_shortest 50;

grunt> dump longest\_movies;

8) Name of highest rated movie in each year.

grunt> grouped\_by\_year = GROUP movies BY year;

grunt> describe grouped\_by\_year;

grouped\_by\_year: {group: int,movies: {(id: int,name: chararray,year: int,rating: double,duration: int)}}  
  
grunt> movies\_year\_highrating = FOREACH grouped\_by\_year generate group as year, MAX(movies.rating)as high\_rating;

grunt> describe movies\_year\_highrating;

movies\_year\_highrating: {year: int,high\_rating: double}  
  
grunt> join\_title = join movies\_year\_highrating by (year,high\_rating), movies by (year,rating);

grunt> describe join\_title;

join\_title: {movies\_year\_highrating::year: int,movies\_year\_highrating::high\_rating: double,movies::id: int,movies::name: chararray,movies::year: int,movies::rating: double,movies::duration: int}

grunt> best\_movies = FOREACH join\_title generate $0 as year,$3 as title, $1 as rating;

grunt> describe best\_movies;

best\_movies: {year: int,title: chararray,rating: double}

grunt> dump best\_movies;