

## Mini Project 2: Cloud computing: Relational Data Analysis with Pig and Spark

### 1.1 Pig and Pig Latin

After you download, unzip, and copy the pig binary to a directory, you need to check the following items.

- Since you will be working with the local mode "pig -x local", you can skip the setup of the pig-specific environment variables.
- work through [the "Getting Start" tutorial](#) and make sure you are familiar with the basic environment.
- Read the lecture slides and the online materials [the Pig Latin Language](#). Make sure you fully understand the usage of the frequently used operations: LOAD, STORE, DUMP, FILTER, FOREACH-GENERATE, GROUP, JOIN, DISTINCT, ORDER, LIMIT, which might be used in solving the problems in the project.

### 1.2 Spark and Spark SQL

After you download, unzip, and copy the spark binary to a directory, you need to do the following tasks

- follow the instructions in the lecture slides to config the spark system for the single-node mode.
- Review the lecture slides on Spark essentials and [the online tutorial](#). Please focus on the Python APIs for a less steep learning curve. We have studied the Spark's functional programming style in the class. Students who did not attend the class may need more time to self-study the related topics.

Now, answer the following questions:

**Question 1.1** Assume the input file is a text document. Please explain what the following piece of code does, line by line. What does this piece of code actually do? (hint: if you are not familiar with the syntax, try the code in pig shell and dump the intermediate results)

```
A = load './input.txt';
```

Ans: Loads input from the text file.

```
(Hello This is my First Line)
(Hello This is my Second Line)
(Hello This is my Third Line)
(Hello This is my Fourth Line)
(Hello This is my Fifth Line)
```

```
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as token;
```

Ans: Each and every word in each and every line is flattened ( Merging nested arrays to a single array) and returns all the words as a single list B.

(Hello)  
(This)  
(is)  
(my)  
(First)  
(Line)  
(Hello)  
(This)  
(is)  
(my)  
(Second)  
(Line)  
(Hello)  
(This)  
(is)  
(my)  
(Third)  
(Line)  
(Hello)  
(This)  
(is)  
(my)  
(Fourth)  
(Line)  
(Hello)  
(This)  
(is)  
(my)  
(Fifth)  
(Line)

C = **group** B **by token**;

Ans: Similar words are grouped with key repeating word, value, list of repeating characters

(is,{(is),(is),(is),(is),(is)})  
(my,{(my),(my),(my),(my),(my)})  
(Line,{(Line),(Line),(Line),(Line),(Line)})  
(This,{(This),(This),(This),(This),(This)})  
(Fifth,{(Fifth)})  
(First,{(First)})  
(Hello,{(Hello),(Hello),(Hello),(Hello),(Hello)})  
(Third,{(Third)})  
(Fourth,{(Fourth)})  
(Second,{(Second)})

D = **foreach** C **generate** group, COUNT(B);

Ans: Count of occurrence of each word is generated.

```
2010-10-19 22:34:33,330 I
```

```
(is,5)
(my,5)
(Line,5)
(This,5)
(Fifth,1)
(First,1)
(Hello,5)
(Third,1)
(Fourth,1)
(Second,1)
_
```

**store** D **into** './output';

Ans: Output of D is stored in file.

**Question 1.2** Convert the code in 1.1 to Spark (python), and paste the code here.

```
>>> textFile = sc.textFile("input.txt")

>>> textFile=textFile.flatMap(lambda line: line.split())

>>> textFile= textFile.groupBy(lambda word: word)

>>> textFile=textFile.map(lambda (x,iterator):(x,len(iterator)))

>>> result=textFile.collect()
```

**// Write this result to a file. ask keke**

**Question 1.3** Assume the input file is a text file. Please explain the following piece of PySpark code, line by line. What does this piece of code actually do?

```
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
```

Ans: Each and every line is stored in an array.

```
[u'1 100', u'2 200', u'3 300', u'4 400', u'5 500', u'6 600', u'7 700', u'8 800', u'9 900', u'10 1000']
```

```
A = lines.flatMap(lambda x: x.split(' '))
```

Ans: Each and every word is split and flattened.

```
[u'1', u'100', u'2', u'200',  
u'3', u'300', u'4', u'400', u'5', u'500', u'6', u'600', u'7', u'700', u'8', u'800', u'9', u'900', u'10',  
u'1000']
```

```
B = A.map(lambda x: (int(x), 1))
```

1 is appended to each and every number present in the list.

```
[(1, 1), (100, 1), (2, 1), (200, 1), (3, 1), (300, 1), (4, 1), (400, 1), (5, 1), (500, 1), (6, 1), (600, 1), (7,  
1), (700, 1), (8, 1), (800, 1), (9, 1), (900, 1), (10, 1), (1000, 1)]
```

```
C = B.sortByKey()
```

List is sorted by key

```
[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (100, 1), (200, 1), (300, 1),  
(400, 1), (500, 1), (600, 1), (700, 1), (800, 1), (900, 1), (1000, 1)]
```

**Question 1.4** How much time did you spend on the task 1.1-1.3? (in hours)

3 hours

**Question 1.5** How useful are these tasks to your understanding of the Pig Latin language and Spark APIs? (very useful, somewhat useful, not useful)

Very useful

---

## Part 2: Analyzing Book Sales Records

In this task, you will use Pig and Spark SQL to solve the analytic problems for a set of linked tables. Now we have three data files with the following schema: customer(cid, name, age, city, sex), book(isbn, name), and purchase(year, cid, isbn, seller, price), where purchase.cid is the foreign key to customer.cid and purchase.isbn is the foreign key to book.isbn. The fields in the dataset are separated by "\t". You should assign the schema, i.e., field names and and types, to the fields when you load the data.

The pig local mode is recommended for better performance:

```
pig -x local [script_file]
```

and Spark Python should be used for the Spark SQL question.

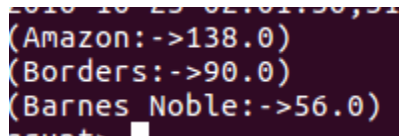
The code you give for the following questions should be well commented.

**Question 2.1** How much did each seller earn? Paste the Pig code and the query answer here.

```
A = Load '/home/meena/Downloads/pig-0.16.0/script/input/purchase' AS (year:int,
cid:chararray, isbn:chararray, seller:chararray, price:float);
B = GROUP A BY seller; --Group by seller
```

```
/*(Amazon,{(2008,C3,B3,Amazon,28.0),(2001,C1,B2,Amazon,20.0),(1999,C1,B1,Amazon,90.0)}}
(Borders,{(2009,C2,B1,Borders,90.0)}}
(Barnes Noble,{(2010,C4,B3,Barnes Noble,26.0),(2008,C2,B2,Barnes Noble,30.0)}})*/
```

```
C = FOREACH B GENERATE CONCAT(CONCAT(group,'->'),(chararray)SUM(A.price));
DUMP C; --Generate output
```



```
(Amazon:->138.0)
(Borders:->90.0)
(Barnes Noble:->56.0)
```

**Question 2.2** Find the names of the books that Amazon gives the lowest price among all sellers.  
Paste the Pig code and the query answer here.

**Ans:**

```
purchase_inp = LOAD '/home/meena/Downloads/pig-0.16.0/script/input/purchase' AS
(year:int, cid:chararray, isbn:chararray, seller:chararray, price:float); -- Load input
```

```
book_inp = LOAD '/home/meena/Downloads/pig-0.16.0/script/input/book' AS (isbn:chararray,
name:chararray);
```

```
group_book = GROUP purchase_inp BY isbn; -- Group by isbn
```

```
/*(B1,{(2009,C2,B1,Borders,90.0),(1999,C1,B1,Amazon,90.0)}}
(B2,{(2008,C2,B2,Barnes Noble,30.0),(2001,C1,B2,Amazon,20.0)}}
(B3,{(2010,C4,B3,Barnes Noble,26.0),(2008,C3,B3,Amazon,28.0)}})*/
```

```
cost_least = FOREACH group_book GENERATE group as isbn, MIN(purchase_inp.price) as
minimum_purchase; -- Gets the minimum price of each book
```

```
/*(B1,90.0)
(B2,20.0)
(B3,26.0)*/
```

```
merge_costs = JOIN cost_least by minimum_purchase, purchase_inp by price; -- Join minimum
price with purchase_inp
```

```
/*(B2,20.0,2001,C1,B2,Amazon,20.0)
(B3,26.0,2010,C4,B3,Barnes Noble,26.0)
(B1,90.0,2009,C2,B1,Borders,90.0)
(B1,90.0,1999,C1,B1,Amazon,90.0)*/
```

select\_amazon= FILTER merge\_costs BY purchase\_inp::seller == 'Amazon'; -- **Filter the merge\_costs table by seller = 'Amazon'**

```
/*(B2,20.0,2001,C1,B2,Amazon,20.0)
(B1,90.0,1999,C1,B1,Amazon,90.0)*/
```

list\_books= FOREACH select\_amazon GENERATE cost\_least::isbn as isbn; -- **Get the books isbn that Amazon sells at low price**

```
/*(B2)
(B1)*/
```

merge\_books = JOIN list\_books by isbn, book\_inp by isbn; -- **Join the book isbn with book name**

```
/*(B1,B1,Novel)
(B2,B2,Drama)*/
```

out = FOREACH merge\_books GENERATE book\_inp::name; -- **Get the output**

```
(Novel)
(Drama)
```

**Question 2.3** Who also bought **ALL** the books that Harry bought? Paste the **Pig** and **Spark SQL** code, and the query answer here.

**PIG**

purchase\_inp = LOAD '/home/meena/Downloads/pig-0.16.0/script/input/purchase' AS (year:int, cid:chararray, isbn:chararray, seller:chararray, price:float);

customer\_inp = LOAD '/home/meena/Downloads/pig-0.16.0/script/input/customer' AS (cid:chararray, cname:chararray);

customer = FILTER customer\_inp by cname == 'Harry Smith'; -- **Gives the cid for Harry**

```
/*(C2,Harry Smith)*/
```

customer\_id = FOREACH customer GENERATE cid AS cid;

```
/*(C2)*/
```

```
join_purchase = JOIN customer_id by cid, purchase_inp by cid;
```

```
/*(C2,2009,C2,B1,Borders,90.0)
(C2,2008,C2,B2,Barnes Noble,30.0)*/
```

```
books = FOREACH join_purchase GENERATE purchase_inp::isbn as isbn; -- Generate isbn of books that Harry has bought
```

```
/*(B1)
(B2)*/
```

```
join_books = JOIN books by isbn, purchase_inp by isbn; -- JOIN the books with that of purchase_inp
```

```
/*(B1,2009,C2,B1,Borders,90.0)
(B1,1999,C1,B1,Amazon,90.0)
(B2,2008,C2,B2,Barnes Noble,30.0)
(B2,2001,C1,B2,Amazon,20.0)*/
```

```
all_cid = FOREACH join_books GENERATE purchase_inp::cid as cid; -- generate cid for books that harry bought
```

```
/*(C2)
(C1)
(C2)
(C1)*/
```

```
group_by_customerid= GROUP all_cid BY cid;
```

```
/*(C1,{{(C1),(C1)}})
(C2,{{(C2),(C2)}})*/
```

```
customer_cid= FOREACH group_by_customerid GENERATE group AS cid;
```

```
/*(C1)
(C2)*/
```

```
join_customer_inp = JOIN customer_cid by cid, customer_inp by cid;
```

```
/*(C1,C1,Jackie Chan)
(C2,C2,Harry Smith)*/
```

```
customer_name = FOREACH join_customer_inp GENERATE customer_inp::cname as cname;
```

```
/*(Jackie Chan)
(Harry Smith)*/
```

```
out = FILTER customer_name BY cname != 'Harry Smith';
```

```
dump out;
```

```
(Jackie Chan)
```

## SPARK SQL

```
purchase_inp=sc.textFile("/home/meena/Downloads/pig-
0.16.0/script/input/purchase").map(lambda line:line.split("\t")) --Load a text file and convert
each line to a row
```

```
>>> purchase_inp.collect()
[[('1999', 'C1', 'B1', 'Amazon', '90'), ('2001', 'C1', 'B2', 'Amazon', '20'), ('2008', 'C2', 'B2', 'Barnes Noble', '30'), ('2008', 'C3', 'B3', 'Amazon', '28'), ('2009', 'C2', 'B1', 'Borders', '90'), ('2010', 'C4', 'B3', 'Barnes Noble', '26')]]
```

```
purchase_inp=purchase_inp.map(lambda p: Row(year=int(p[0]), cid=p[1], isbn=p[2], seller=p[3],
price=int(p[4]))) --Define the schema
```

```
>>> purchase_inp.collect()
[Row(cid='C1', isbn='B1', price=90, seller='Amazon', year=1999), Row(cid='C1', isbn='B2', price=20, seller='Amazon', year=2001), Row(cid='C2', isbn='B2', price=30, seller='Barnes Noble', year=2008), Row(cid='C3', isbn='B3', price=28, seller='Amazon', year=2008), Row(cid='C2', isbn='B1', price=90, seller='Borders', year=2009), Row(cid='C4', isbn='B3', price=26, seller='Barnes Noble', year=2010)]
```

```
customer_inp=sc.textFile("/home/meena/Downloads/pig-
0.16.0/script/input/customer").map(lambda line:line.split("\t"))
```

```
customer_inp=customer_inp.map(lambda p: Row(cid=p[0], name=p[1]))
```

```
customer=customer_inp.filter(lambda p: p.name=='Harry Smith') --Filter the customer table for
Harry Smith
```

```
>>> customer.collect()
[Row(cid='C2', name='Harry Smith')]
```

```
purchase_inp=spark.createDataFrame(purchase_inp) --Register Dataframe as table
```



```
>>> purchase_inp.show()
+---+-----+-----+-----+
|cid|isbn|price|      seller|year|
+---+-----+-----+-----+
| C1|  B1|  90|    Amazon|1999|
| C1|  B2|  20|    Amazon|2001|
| C2|  B2|  30|Barnes Noble|2008|
| C3|  B3|  28|    Amazon|2008|
| C2|  B1|  90|    Borders|2009|
| C4|  B3|  26|Barnes Noble|2010|
+---+-----+-----+-----+
```

```
customer_inp=spark.createDataFrame(customer_inp)
```

```
customer_inpschema=spark.createDataFrame(customer)
```

```
books=purchase_inp.join(customer_inpschema,
purchase_inp.cid==customer_inpschema.cid).drop(purchase_inp.cid) - Join purchase table
with Harry's customer id
```

```
>>> books.show()
+---+-----+-----+-----+-----+-----+
|isbn|price|      seller|year|cid|      name|
+---+-----+-----+-----+-----+-----+
|  B2|   30|Barnes Noble|2008| C2|Harry Smith|
|  B1|   90|    Borders|2009| C2|Harry Smith|
+---+-----+-----+-----+-----+-----+
```

```
books=books.alias("books")
```

```
group_customer_id=books.join(purchase_inp,books.isbn==purchase_inp.isbn).drop(books.isbn)
.drop(books.cid).drop(books.price).drop(books.seller).drop(books.year).drop(books.name) - -
Join the purchase table and books table with isbn
```

```
>>> group_customer_id.show()
+---+-----+-----+-----+-----+
|cid|isbn|price|      seller|year|
+---+-----+-----+-----+-----+
| C1|  B2|   20|    Amazon|2001|
| C2|  B2|   30|Barnes Noble|2008|
| C1|  B1|   90|    Amazon|1999|
| C2|  B1|   90|    Borders|2009|
+---+-----+-----+-----+-----+
```

```
group_customer_name=group_customer_id.join(customer_inp,group_customer_id.cid==custo
mer_inp.cid).drop(customer_inp.cid) --Join group_customer_id and customer_inp with cid
```

```
>>> group_customer_name.show()
+-----+
|cid|isbn|price|seller|year|name|
+-----+
| C1| B2| 20| Amazon|2001|Jackie Chan|
| C1| B1| 90| Amazon|1999|Jackie Chan|
| C2| B2| 30| Barnes Noble|2008|Harry Smith|
| C2| B1| 90| Borders|2009|Harry Smith|
+-----+
```

customer\_name=group\_customer\_name.filter(group\_customer\_name.name!='Harry Smith') —  
**Filter group\_customer\_name for name not equal to Harry Smith**

```
>>> customer_name.show()
+-----+
|cid|isbn|price|seller|year|name|
+-----+
| C1| B2| 20| Amazon|2001|Jackie Chan|
| C1| B1| 90| Amazon|1999|Jackie Chan|
+-----+
```

delete\_dup=customer\_name.dropDuplicates(['name']) — **Delete duplicate names**

```
>>> delete_dup.show()
+-----+
|cid|isbn|price|seller|year|name|
+-----+
| C1| B2| 20| Amazon|2001|Jackie Chan|
+-----+
```

final\_output=delete\_dup.select(delete\_dup.name)

```
>>> final_output.show()
+-----+
|name|
+-----+
|Jackie Chan|
+-----+
```

**Question 2.4** How much time did you spend on the tasks 2.1-2.3? (hours)

48 hours

**Question 2.5** How useful are these tasks to your understanding of Pig and Spark programming?  
 (very useful, somewhat useful, not useful)

Very useful

**Final Survey Questions**

**Question 3.1** Your level of interest in this lab exercise (high, average, low);

High

**Question 3.2** How challenging is this lab exercise? (high, average, low);

Average

**Question 3.3** How valuable is this lab as a part of the course (high, average, low).

High

**Question 3.4** Are the supporting materials and lectures helpful for you to finish the project?  
(very helpful, somewhat helpful, not helpful);

Somewhat helpful

**Question 3.5** How much time in total did you spend in completing the lab exercise;

51 hours

**Question 3.6** Do you feel confident on applying the skills learned in the lab to solve other problems?

Yes