ISIT312 Big Data Management

Session 4, 2021

Exercise 7

Using Apache Pig Latin

In this exercise you will learn how to use Apache Pig Latin for processing of the data files stored in HDFS.

Be careful when copying the Linux commands in this document to your working Terminal, because it is error-prone. Maybe you should type those commands by yourself.

Prologue

Login to your system and start VirtualBox.

When ready start a virtual machine `ISIT312-BigDataVM-07-SEP-2020`.

(1) How to start Hadoop?

Open a new Terminal window and start Hadoop in the following way.

```
$HADOOP_HOME/sbin/start-all.sh
```

When using Pig we have to additionally start History Server. Use the following command in a Terminal Window to start History Server.

```
$HADOOP_HOME/sbin/m*sh start historyserver
```

Create a text file `orders.txt` with the following contents:

```
bolt,James,200,2016,01,01
bolt,Peter,100,2017,01,30
bolt,Bob,300,2018,05,23
screw,James,20,2017,05,11
screw,Alice,55,2018,01,01
nut,Alice,23,2018,03,16
washer,James,45,2016,04,24
washer,Peter,100,2016,05,12
bolt,James,200,2018,01,05
bolt,Peter,100,2018,01,05
bolt,James,,2018,01,01
```

And load the file into HDFS into a location `/user/bigdata` .

(2) How to start Grunt Shell ?

To start Grunt Shell process the following command in a Terminal window.

```
$PIG_HOME/bin/pig
```

Apache Pig command line interface (Grunt) should open a new line with a prompt `grunt>`.

As usual, we start from asking about help. Process `help` command at `grunt>` prompt. The outcomes are listed in Appendix A at the end of this document.

It is possible to type and to process Pig commands directly at `grunt>` prompt. For few simple commands it is probably all right to do so. However, the longer sequences to commands require application of Pig script.

(3) How to process a sequence of Pig commands?

It is possible to process a sequence of Pig commands separated with '`;`' as a value of `-e` switch of `pig` command. For example, it is possible to load into Pig container orders the contents of a file `orders.txt` located in HDFS and list the file with `dump` command as below.

```
$PIG_HOME/bin/pig -e "orders = load '/user/bigdata/orders.txt'
using PigStorage(','); dump orders;"
```

Of course, the best solution is to create Pig script file and process the file. Create a text file `script.pig` put into it the following lines.

```
orders = /user/bigdata/orders.txt' using PigStorage(',');
dump orders;
```

Then, save the file and process the following command at `$` prompt in a Terminal window.

```
$PIG_HOME/bin/pig -f script.pig
```

Note, that a clause `PigStorage(',')` determines a separator between the values in the rows of input data file. By default a separator is `TAB`. In our case the values are separated with a comma.

(4) How to assign the names and types to columns in Pig data container ?

Process the following command at `grunt>` prompt. A clause `as ( … )` determines the names of columns and the types of columns.

```
orders = load '/user/bigdata/orders.txt' using PigStorage(',')
as
(item:chararray,customer:chararray,quantity:int,year:int,month:i
nt,day:int);
```

To verify the results, process a command `describe`.

```
describe orders;
```

(5) How to load a map into Pig storage ?

Create a text file `keyvalue.txt` that contains the following lines.

```
[first-name#James,last-name#Bond,city#London,country#UK]
[first-name#Harry,last-name#Potter,city#Avondale,country#UK]
```

Next, load a file `keyvalue.txt` into HDFS folder `/user/bigdata`.

Then, process the following commands at `grunt>` prompt.

```
keyvalue = load '/user/bigdata/keyvalue.txt'as
(department:chararray,personal:map[]);
describe keyvalue;
dump keyvalue;
```

A data container `keyvalue` contains two maps earlier loaded into HDFS.

(6) How to load a bag with tuples into Pig storage ?

Create a text file `hobbies.txt` with the following rows.

```
James,({painting},{swimming})
Harry,({cooking})
Robin,({})
```

The file contains information about the first names of people and their hobbies.

Next load a file `hobbies.txt` into HDFS folder `/user/bigdata`.

Then process the following commands at `grunt>` prompt.

```
hobbies = load '/user/bigdata/hobbies.txt' as

(firstname:chararray,hobbies:bag{t:(hobby:chararray)});
dump hobbies;
describe hobbies;
```

Note, that a name of a bag `t` must be included in a specification of bag structure. It is also possible to nest within a bag not only sets of values but also sets of tuples.

Create a text file `nested.txt` with the following contents.

```
James,({Ferrari,xyz123}{Honda,pkr856})
Harry,({Rolls Royce,xxx666})
```

Next load a file `nested.txt` into HDFS folder `/user/bigdata`.

Then process the following commands at `grunt>` prompt.

```
nested = load '/user/bigdata/nested.txt' as
(firstname:chararray,cars:bag{t:(manufacturer:chararray, rego:chararray)});
dump nested;
describe nested;
```

(7) How to compute projections of a data container ?

We start from `describe` command to refresh our memory on a structure of `orders` container create in step 3.

```
  describe orders;
```

Assume that we would like to create a container `items` with single column `item` extracted from a container `orders`. Process the following commands at `grunt>` prompt.

```
  items = foreach orders generate item;
  dump items;
```

It is possible to remove duplicates with Remove duplications with distinct operation.

```
distinctitems = distinct items;
dump distinctitems;
describe distinctitems;
```

Projection on many columns

```
  dates = foreach orders generate day, month, year;
  dump dates;
```

(8) How to perform selections from data container ?

A `filter` command can be used to select all orders where quantity is greater than 100;

```
  biggerorders = filter orders by quantity > 100;
  dump biggerorders;
```

A useful example is to check the rows for nulls.

```
  nulls = filter orders by quantity is null;
  dump nulls;
```

(9) How to split data containers ?

Sometimes we would like to save the rows filtered from a data container into several other containers. For example, we split a container `orders` into a container `orders2018` that contains orders submitted in 2018 and a container `olderorders` that contains other orders.

```
  split orders into
    orders2018 if year==2018,
    olderorders otherwise;
  dump orders2018;
```

(10) How to compute an inner join ?

Create a data set `items.txt` with the following contents

```
  bolt,2.23
  screw,3.5
```

```
nut,1.25
washer,2.5
nail,0.4
fastener,4.1
pin,10.05
coupler,9.95
```

and load it into HDFS into a folder `/user/bigdata` .

Next, create a new data container `items` in the following way.

```
items = load '/user/bigdata/items.txt' using PigStorage(',') as
             (item:chararray,price:float);
dump items;
```

Now we implement an inner join operation to find the prices of all ordered products.

```
orders_join_items = join orders by item, items by item;
describe orders_join_items;
dump orders_join_items;
```

(11) How to implement left outer join ?

A left outer join operation joins all `items` with `orders` and includes into the result the items that have not been ordered so far extended with nulls in all columns from a container `orders`. Process the following commands.

```
leftouter = join items by item left outer, orders by item;
dump leftouter;
describe leftouter;
```

The results saved in `leftouter` data container can be used to find the names of items that have not been ordered yet, i.e. it is nothing else but implementation of *antijoin* operation. *Antijoin* operation, in contrast to join operation, finds all rows from the left argument of the operation that cannot be joined with the rows from the right argument of the operation. Process the following commands to find all items that have not been ordered yet.

```
notordered = filter leftouter by orders::item is null;
dump notordered;
```

(12) How to implement non-equi join ?

An inner join operation computed in a step 10 assumes that the rows are connect only when a value in a column `item` in a container `items` is the same as a value in a column `item` in a container `orders`. It is so called equi join. It is possible to implement a join operation that joins the rows from two containers that satisfy any condition different from equality condition. For example, find all pairs of items such that the first element in each pair has a price higher than the second element. Process the following sequence of commands and verify the results after each step.

```
newitems = load '/user/bigdata/items.txt' using PigStorage(',')
           as (item:chararray,price:float);
```

```
  crossjoin = cross items, newitems;
  describe crossjoin;
  result = filter crossjoin by items::price > newitems::price;
```

(13) How to perform groupings and how to compute aggregation functions ?

An operation `group by` can be used to restructure a container with flat tuples into a container with tuple nested with bags. Assume that we would like to aggregate all orders on the same items into the bags and assign these bags with the ordered item. This can be achieved in the following way.

```
  ordergrp = group orders by item;
```

To find an internal structure of a data container `ordergrp` process a command

```
  describe ordergrp;
```

and later on a command

```
  dump ordergrp;
```

Now, it is possible to count the total number of orders in each bag to get the result identical to `SELECT` with `GROUP BY` clause of SQL.

```
  itemscnt = foreach ordergrp generate group, COUNT(orders.item);
  dump itemsscnt;
```

(14) How to process `CUBE` and `ROLLUP` operators ?

`CUBE` operator performs grouping over all subsets of a give set of values and saves the results in a bag.

Process the following commands.

```
  ordcube = cube orders by CUBE(item,name);
  describe ordcube;
  dump ordcube;
```

Like with `group` operation it is possible to apply an aggregation function to the results.

```
  cntcube = foreach ordcube generate group, COUNT(cube.item);
  dump cntcube;
```

In the same way it is possible to process `ROLLUP` operator.

```
ordrollup = cube orders by ROLLUP(item,name);
dump ordrollup;
describe ordrollup;

cntrollup = foreach ordrollup generate group, COUNT(cube.item);
dump cntrollup;
```

(15) How to unnest (flatten) data bags ?

Consider a data container `ordergrp` created in a step (13).

```
describe ordergrp;

ordergrp: {group: chararray,orders: {(item: chararray,
    name: chararray,quantity: int,year: int,month: int,day: int)}}
```

An operation `flatten` can be used to "ungroup" a nested structure. Process the following commands.

```
orderunnest = foreach ordergrp generate flatten (orders);
describe orderunnest;
dump orderunnest;
```

(16) How to save the results in HDFS ?

To save the contents of a data container created by processing Pig Latin commands use a command `store`.

```
  store ordergrp into '/user/bigdata/orders' using
  PigStorage(',');
  store ordcube into '/user/bigdata/ordcube' using
  PigStorage('|');
```

Next, in a new terminal window use the following command to list the contents of HDFS.

```
$HADOOP_HOME/bin/Hadoop fs -ls /user/bigdata/orders
$HADOOP_HOME/bin/Hadoop fs -cat /user/bigdata/orders/part-m-00000
```

(17) How to process Hadoop commands in front of `grunt>` prompt ?

To list the contents of HDFs process the following `fs` command at `grunt>` prompt.

```
fs -cat /user/bigdata/orders/part-m-00000
```

Appendix A

The outcomes of `help` command.

```
<pig latin statement>; - See the PigLatin manual for details:
http://hadoop.apache.org/pig
File system commands:
    fs <fs arguments> - Equivalent to Hadoop dfs command:
http://hadoop.apache.org/common/docs/current/hdfs_shell.html
Diagnostic commands:
    describe <alias>[::<alias] - Show the schema for the alias. Inner
aliases can be described as A::B.
    explain [-script <pigscript>] [-out <path>] [-brief] [-dot|-xml] [-
param <param_name>=<param_value>]
        [-param_file <file_name>] [<alias>] - Show the execution plan to
compute the alias or for entire script.
        -script - Explain the entire script.
        -out - Store the output into directory rather than print to
stdout.
        -brief - Don't expand nested plans (presenting a smaller graph
for overview).
        -dot - Generate the output in .dot format. Default is text
format.
        -xml - Generate the output in .xml format. Default is text
format.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        alias - Alias to explain.
    dump <alias> - Compute the alias and writes the results to stdout.
Utility Commands:
    exec [-param <param_name>=param_value] [-param_file <file_name>]
<script> -
        Execute the script with access to grunt environment including
aliases.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        script - Script to be executed.
    run [-param <param_name>=param_value] [-param_file <file_name>]
<script> -
        Execute the script with access to grunt environment.
        -param <param_name - See parameter substitution for details.
        -param_file <file_name> - See parameter substitution for details.
        script - Script to be executed.
    sh  <shell command> - Invoke a shell command.
    kill <job_id> - Kill the hadoop job specified by the hadoop job id.
    set <key> <value> - Provide execution parameters to Pig. Keys and
values are case sensitive.
        The following keys are supported:
        default_parallel - Script-level reduce parallelism. Basic input
size heuristics used by default.
        debug - Set debug on or off. Default is off.
        job.name - Single-quoted name for jobs. Default is
PigLatin:<script name>
        job.priority - Priority for jobs. Values: very_low, low, normal,
high, very_high. Default is normal
        stream.skippath - String that contains the path. This is used by
streaming.
        any hadoop property.
    help - Display this message.
    history [-n] - Display the list statements in cache.
        -n Hide line numbers.
    quit - Quit the grunt shell.For a good start use a command help to
get a pretty comprehensive help from HBase command Line Interface (CLI).
A complete printout of help is listed at the end of this document.
```

End of exercise 7