Goel, Ajay: Nu Id (001897443)

INFO 7250
Engineering Big-Data Systems
Summer Full 2019
Assignment 5

PART 3 - Execute 2 HBase commands from each of the 6 groups (Total 12 commands), and place the screenshots into a word file, and upload to BlackBoard.

General HBase shell commands

1. Status:
   Show cluster status. Can be 'summary', 'simple', or 'detailed'. The default is 'summary'.

2. Version
   Output this HBase version**Usage:**

```
hbase(main):006:0*
hbase(main):007:0* version
1.3.4, r5d443750f65c9b17df23867964f48bbd07f9267d, Mon Apr 15 02:17:47 UTC 2019
```

3. Whoami

```
hbase(main):008:0> whoami
ajaygoel (auth:SIMPLE)
    groups: staff, everyone, localaccounts, _appserverusr, admin, _appserveradm, _lpadmin, _app
ple.access_ssh, com.apple.sharepoint.group.1
```

Tables Management commands

4. Create
   Create table; pass table name, a dictionary of specifications per column family, and optionally a dictionary of table configuration.

```
create 'emp', 'personal data', 'professional data'
```

```
list
```

```
hbase(main):001:0> create 'emp', 'personal data', 'professional data'
0 row(s) in 2.4920 seconds

=> Hbase::Table - emp
hbase(main):002:0> list
TABLE
emp
log
2 row(s) in 0.0250 seconds

=> ["emp", "log"]
hbase(main):003:0>
```

5. Describe
   Describe the table created

```
=> ["emp", "log"]
hbase(main):003:0> describe 'emp'
Table emp is ENABLED
emp
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal data', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL
> '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
{NAME => 'professional data', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE',
NS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.0310 seconds

hbase(main):004:0>
```

6. Exists
   Does the table exist with the given name.

```
hbase(main):004:0>
hbase(main):005:0* exists 'emp'
Table emp does exist
0 row(s) in 0.0060 seconds

hbase(main):006:0>
```

Data Manipulation Commands

7. put data :
   Inserting the data

```
hbase(main):005:0> put 'emp','1','personal data:name','raju'
0 row(s) in 0.6600 seconds
```

```
hbase(main):006:0> put 'emp','1','personal data:city','hyderabad'
0 row(s) in 0.0410 seconds
hbase(main):007:0> put 'emp','1','professional
data:designation','manager'
0 row(s) in 0.0240 seconds
hbase(main):007:0> put 'emp','1','professional data:salary','50000'
0 row(s) in 0.0240 seconds
```

```
 => ["emp", "log"]
hbase(main):005:0> emp
 NameError: undefined local variable or method `emp' for #<Object:0x707ca986>

 hbase(main):006:0> delete 'emp', '1', 'personal data:city',
hbase(main):007:0* 1417521848375
 0 row(s) in 0.1080 seconds

hbase(main):008:0> descibe 'emp'
 NoMethodError: undefined method `descibe' for #<Object:0x707ca986>

hbase(main):009:0> scan 'emp'
 ROW                                                COLUMN+CELL
 0 row(s) in 0.0230 seconds

hbase(main):010:0> descibe 'emp'
 NoMethodError: undefined method `descibe' for #<Object:0x707ca986>

hbase(main):011:0> describe 'emp'
 Table emp is ENABLED
 emp
```

8. Scan ’emp’

```
hbase(main):017:0> scan 'emp'
 ROW                                                COLUMN+CELL
  1                                                 column=personal data:city, timestamp=1563855638431, value=hyderabad
  1                                                 column=personal data:name, timestamp=1563855624773, value=raju
  1                                                 column=professional data:salary, timestamp=1563855651929, value=50000
1 row(s) in 0.0270 seconds
```

HBASE Surgery Tools

9. Balancer:

   Trigger the cluster balancer. Returns true if balancer ran and was able to tell the region servers to unassign all the regions to balance.

```
hbase(main):001:0> balancer
 true
 0 row(s) in 0.1530 seconds
```

10. Balance_switch:

    Enable/Disable balancer. Returns previous balancer state.

```
Display all 493 possibilities? (y or n)
hbase(main):002:0> balance_switch true
true
0 row(s) in 0.0250 seconds

hbase(main):003:0> balance_switch false
true
0 row(s) in 0.0120 seconds

hbase(main):004:0>
```

Cluster Replication Tools:

11. Add_peer

```
Display all 493 possibilities? (y or n)
hbase(main):004:0> add_peer '1','localhost:2181:/hbase'
0 row(s) in 0.0380 seconds
```

12. List_peer

```
hbase(main):005:0> list_peers
 PEER_ID CLUSTER_KEY STATE TABLE_CFS
 1 localhost:2181:/hbase ENABLED
1 row(s) in 0.0310 seconds
```

Security Tools:

13. Grant

    The **grant** command grants specific rights such as read, write, execute, and admin on a table to a certain user.

```
grant 'Tutorialspoint', 'RWXCA'
```

14. Revoke:

    The **revoke** command is used to revoke a user's access rights of a table.

```
revoke 'Tutorialspoint'
```

15. User_permission

    This command is used to list all the permissions for a particular table. The syntax of **user_permission**

```
user_permission 'emp'
```

```
hbase(main):027:1> grant 'Tutorialspoint', 'RWXCA'
hbase(main):028:1> revoke 'Tutorialspoint'
hbase(main):029:1> user_permission 'emp'
 hbase(main):030:1>
```

PART 4

Building a High-Level Dataflow System on the top of Map-Reduce: The Pig Experience

Introduction:

Pig compiles these dataflow programs, which are written in a language called Pig Latin, into sets of Hadoop Map- Reduce jobs, and coordinates their execution. The Pig system takes a Pig Latin program as input, compiles it into one or more Map-Reduce jobs, and then exe- cutes those jobs on a given Hadoop cluster.

Pig allows three modes of user interaction:

1. Interactive mode: In this mode, the user is presented with an interactive shell (called Grunt), which accepts Pig commands.

2. Batch mode: In this mode, a user submits a pre- written script containing a series of Pig commands, typically ending with STORE. The semantics are identical to interactive mode.

3. Embedded mode: Pig is also provided as a Java library allowing Pig Latin commands to be submitted via method invocations from a Java program.

In interactive mode, two commands are available to help the user reason about the program she is using or creating: DESCRIBE and ILLUSTRATE. The DESCRIBE command displays the schema of a variable (e.g. DESCRIBE urls, DESCRIBE bigGroups). The ILLUSTRATE command displays a small amount of example data for a variable and the variables in its derivation tree, to give a more concrete illustration of the program semantics. Regardless of the mode of execution used, a Pig program goes through a series of transformation steps before being executed

The first step is parsing. The parser verifies that the pro- gram is syntactically correct and that all referenced variables are defined. The parser also performs type checking and schema inference (see Section 3). Other checks, such as verifying the ability to instantiate classes corresponding to user-defined functions and confirming the existence of streaming executables referenced by the user’s program, also occur in this phase. The

output of the parser is a canonical logical plan with a one-to-one correspondence between Pig Latin statements and logical operators, arranged in a directed acyclic graph (DAG).

Type System and Type Inference

Pig has a nested data model, thus supporting complex, non-normalized data. Standard scalar types of int, long, double, and char array (string) are supported. Pig also supports a byte array type that represents a collection of uninterpreted bytes. The type byte array is also used to facilitate unknown data types and lazy conversion of types

Pig supports three complex types: map, tuple, and bag. map is an associative array, tuple is an ordered list of data elements, bag is a collection of tuples. It uses tabs to delimit data values and carriage returns to delimit tuples, and left/right delimiters like { } to encode nested complex types.

Type Declaration

The first option is that no data types are declared. In this case the default is to treat all fields as bytearray. The second option for declaring types in Pig is to provide them explicitly as part of the AS clause during the LOAD

Lazy Conversion of Types

When Pig does need to cast a bytearray to another type because the program applies a type-specific operator, it delays that cast to the point where it is actually necessary. However, these casts will not be done when the data is loaded. Instead, they will be done as part of the comparison and division operations, which avoids casting values that are removed by the filter before the result of the cast is used.

Compilation to Map-Reduce

Pig Latin program is translated in a one-to-one fashion to a logical plan. Each operator is annotated with the schema of its output data, with braces indicating a bag of tuples. A Hadoop Map-Reduce job consists of a series of execution stages. The map stage processes the raw input data, one data item at a time, and produces a stream of data items annotated with keys. A subsequent local sort stage orders the data produced by each machine’s map stage by key. The locally-ordered data is then passed to an (optional) combiner stage for partial aggregation by key.

Plan Execution

This section describes the way Pig executes the portion of a physical plan that lies inside a Map or Reduce stage. In a Pig physical plan, the outermost data flow graph always has exactly one sink and makes no use of pause signals. Our SPLIT and MULTIPLEX operators (Section 4.3.1) are special operators that behave the same as regular operators

from the point of view of the outermost data flow graph, but internally they manage nested, branching data flows with buffering and pause signals.

A naive option is to increase the JVM memory size limit beyond the physical memory size and let the virtual memory manager take care of staging data between memory and disk. Our experiments have confirmed the common wisdom that doing so leads to very severe performance degradation. Most memory overflow situations in Pig arise due to materialization of large bags of tuples between and inside operators, and our memory management approach focuses on dealing with large bags.

Pig's memory manager maintains a list of all Pig bags created in the same JVM, using a linked list of Java Weak References. The size of the bag list is controlled in two ways: (1) when the memory manager adds a bag to the list, it looks for dead bags at the head of the list and removes them, (2) when the low-memory handler is activated and the list is sorted by bag size for spilling, all dead bags are removed. New bags cannot be created during the sort and spill operations.

Streaming:

One of the goals of Pig is to allow users to incorporate custom code wherever necessary in the data processing pipeline. User-defined functions (UDFs) provide one avenue for including user code. But UDFs must be written in Java and must conform to Pig's UDF interface.

Special Pig Latin syntax is used to designate a streaming executable, specify the input and output path- ways (including filenames, if applicable), and specify the input and output data formats used by the executable. Due to the asynchronous behavior of the user's executable, a STREAM operator that wraps the executable cannot simply pull tuples synchronously as it does with other operators because it does not know what state the executable is in. An exception was made in our single-threaded execution model to handle STREAM operators. Each STREAM operator creates two additional threads: one for feeding data to the external executable, and one for consuming data from the executable. Incoming and outgoing data is kept in queues.

Performance

We designed a publicly- available benchmark called Pig Mix to measure performance on a regular basis so that the effects of individual code changes on performance could be understood. Pig Mix consists of a suite of Pig programs and associated data sets, and also includes a raw Map-Reduce- level implementation of each of the programs.

The Pig Mix benchmark consists of twelve Pig programs designed to collectively test the above features. For each Pig program, the Pig Mix has a corresponding group of Hadoop Map-Reduce programs. Pig Mix includes a data generator that produces data with similar properties to Yahoo's proprietary data sets that are commonly processed using Pig.

Nova: Continuous Pig / Hadoop Workflows

1. Introduction:

This paper describes a workflow manager developed and deployed at Yahoo called Nova, which pushes continually- arriving data through graphs of Pig programs executing on Hadoop clusters. Given that Pig itself deals with graphs of inter- connected data processing steps, it is natural to ask why one would layer another graph abstraction on top of Pig. It turns out that this two-layer programming model enables key scheduling and data handling capabilities:

- Continuous processing.
- Independent scheduling
- Cross-module optimization
- Manageability features.

What sets Nova apart is its sup- port for stateful continuous processing of evolving data sets. To avoid replicating basic workflow capabilities, Nova is layered on top of Oozie, which handles dependency-aware batch execution of sets of workflow elements, automatic retry and failure management, and many other core features.

2. Abstract Workflow Model

A workflow is a directed graph with two kinds of vertices: tasks and channels. Tasks are processing steps. Channels are data containers. Edges connect tasks to channels and channels to tasks; no edge can go from a task to a task or from a channel to a channel.

Data and Updated Model

A channel's data is divided into blocks, each of which contains a set of data records or record transformations that have been generated by a single task invocation. Blocks may vary in size from kilobytes to terabytes. For the purpose of workflow management, a block is an atomic unit of data. Blocks also constitute atomic units of processing: a task invocation consumes zero or more input blocks and processes them in their entirety; partial processing of blocks is not permitted. Data blocks are immutable, and so as channels accumulate data blocks their space footprint can grow without bound.

There are two types of blocks: base blocks and delta blocks (bases and deltas, for short). A base block contains a complete snapshot of data on a channel as of some point in time. Delta blocks are used in conjunction with incremental processing.

Each channel has associated merge, chain and diff functions. These functions may vary from channel to channel, depending on the type of data and data updates each channel supports.

Task / Data Interface

A task must declare, for each incoming edge from a data channel, its consumption mode: one of all or new. Consumption mode all denotes that each time the task is executed, it is fed a complete snapshot of the data residing on the input channel in the form of a base block. Consumption mode new denotes that each task execution is to be fed just new data that has not been seen in prior invocations, in the form of a delta block $\Delta i \rightarrow j$, where i is the highest sequence number read in the most recent successful execution and j is the highest sequence number available on the channel

Workflow Programming and Scheduling

Workflows are programmed bottom-up, starting with individual task definitions, and then composing them into work- flow fragments called workflowettes. Workflowettes are abstract processing components that are not attached to specific named data channels—instead they have ports to which input and output channels may be connected. The last step is for the user to communicate scheduling requirements, by associating one or more triggers with a bound workflowette.

Data Compaction and Garbage Collection

Nova performs an important data representation optimization called compaction, which memorizes the result of a merge (and chain) operation. Compaction, coupled with garbage collection, has two potential benefits: (1) if delta blocks contain updates and/or deletions, then the compacted data may take up less space than the non-compacted representation; (2) all-mode consumers do not have to merge (as many) delta blocks on the fly.

3. Tying the Model to Pig / Hadoop

Nova relies on a system called Oozie to execute DAGs of Pig Latin scripts, including some important details such as sandboxing the Pig client code, automatically re-trying failed scripts, and capturing and reporting status and error messages. Nova executes a bound workflowette by first associating Pig Latin expressions with each input and output parameter of each of the workflowette's tasks (as described above), and then sending the resulting DAG of parameter-free Pig Latin scripts to Oozie for execution and monitoring. Oozie reports the final status (success or error) back to Nova. If the bound workflowette execution results in an error, Nova erases any output blocks generated during its execution, to achieve atomicity.

File Formats and Schemas

Pig doesn't have a system catalog, and it expects file formats and schemas to be specified in load statements, either via an explicit in-line schema description or via a special "load function" that reads some catalog or self-describing data file and passes the schema in-formation to Pig. Zebra is a self-describing format that comes with such a Pig load

function. Nova supports both manual and Zebra-based file for- mat and schema specification.

4. Workflow Manager Architecture

Most of Nova's modules are part of a Nova server instance process. The modules in a server instance are stateless; they keep their state externally, in a metadata database. A special watchdog module, man- aged via ZooKeeper leader election, detects unresponsive server instances, kills them, starts fresh replacements, and reconfigures the load balancer's routing table as needed.

Summary:

We have described a workflow manager called Nova that supports continuous, large-scale data processing on top of Pig/Hadoop. Nova is a relatively new system, and leaves open many avenues for future work such as:

- Arbitrary workflow nesting, rather than the current, rigidly tiered model. Better schema migration support,

- Scheduling data compaction automatically, based on some optimization formulation such as minimizing total cost while bounding wasted space.
- Investigating H-Base, or a similar BigTable inspired storage system, as the underlying storage and merging infrastructure for data channels.

A) Pig Latin: A Not-so-foreign language for Data Processing

1. Introduction:

A new language called Pig Latin that combines the best of both worlds: high-level declarative querying in the spirit of SQL, and low-level, procedural programming a` la map-reduce. A Pig Latin program is a sequence of steps, much like in a programming language, each of which carries out a single data transformation. This characteristic is immediately appealing to many programmers. At the same time, the transformations carried out in each step are fairly high-level, e.g., filtering, grouping, and aggregation, much like in SQL. The use of such high-level primitives renders low-level manipulations (as required in map-reduce) unnecessary.

2. Features and Motivation:

Data Flow Language and Interoperability:

Pig Latin, a user specifies a sequence of steps where each step specifies only a single, high-level data transformation. Pig is designed to support ad-hoc data analysis. By operating over data residing in external files, and not taking over control over the data, Pig readily interoperates with other applications in the ecosystem.

The reasons that conventional database systems do re- quire importing data into system-managed tables are three- fold: (1) to enable transactional consistency guarantees, (2) to enable efficient point lookups (via physical tuple identifiers), and (3) to curate the data on behalf of the user, and record the schema so that other users can make sense of the data. Pig only supports read-only data analysis workloads, and those workloads tend to be scan-centric, so transactional consistency and index-based lookups are not required.

Nested Data Model

Pig Latin has a flexible, fully nested data model (described in Section 3.1), and allows complex, non-atomic data types such as set, map, and tuple to occur as fields of a table.

UDFs as First-Class Citizens

To accommodate specialized data processing tasks, Pig Latin has extensive support for user-defined functions (UDFs). Essentially all aspects of processing in Pig Latin including grouping, filtering, joining, and per-tuple processing can be customized through the use of UDFs.

The input and output of UDFs in Pig Latin follow our flexible, fully nested data model. Consequently, a UDF to be used in Pig Latin can take non-atomic parameters as input, and also output non-atomic values. Latin has only one type of UDF that can be used in all the constructs such as filtering, grouping, and per-tuple processing. Currently, Pig UDFs are written in Java.

Debugging Environment

Pig comes with a novel interactive debugging environment that generates a concise example data table illustrating the output of each step of the user's program. The example data is carefully chosen to resemble the real data as far as possible and also to fully illustrate the semantics of the program. Moreover, the example data is automatically adjusted as the program evolves. This step-by-step example data can help in detecting errors early (even before the first iteration of running the program on the full data), and also in pinpointing the step that has errors.

Pig comes with a debugging environment called Pig Pen, which creates a side data set automatically, and in a manner that avoids the problems outlined in the previous paragraph. We refer to this dynamically-constructed side data set as a sandbox data set. Pig Pen's sandbox data set generator takes as input a Pig Latin program P consisting of a sequence of n commands, where each command consumes one or more input bags

and produces an output bag. The output of the data set generator is a set of example bags {B1, B2, . . . , Bn}, one corresponding to the output of each command in P.

There are three primary objectives in selecting a sandbox data set:

- Realism. The sandbox data set should be a subset of the actual data set, if possible. If not, then to the extent possible the individual data values should be ones found in the actual data set.
- Conciseness. The example bags should be as small as possible.
- Completeness. The example bags should collectively illustrate the key semantics of each command.

3. Pig Latin

Pig has a rich, yet simple data model consisting of the following four types:

- Atom: An atom contains a simple atomic value such as a string or a number, e.g., ‘alice’.
- Tuple: A tuple is a sequence of fields, each of which can be any of the data types, e.g., (‘alice’, ‘lakers’).
- Bag: A bag is a collection of tuples with possible dupli- cates. The schema of the constituent tuples is flexible, i.e., not all tuples in a bag need to have the same number and type of fields, e.g.,
- Map: A map is a collection of data items, where each item has an associated key through which it can be looked up.

A map is especially useful to model data sets where schemas might change over time.

a) LOAD

queries = LOAD ‘query_log.txt’ USING myLoad() AS (userId, queryString, timestamp);

b) FOR EACH
expanded_queries = FOREACH queries GENERATE userId, expandQuery(queryString);

c) FILTER

real_queries = FILTER queries BY userId neq ‘bot’;

d) COGROUP

Per-tuple processing only takes us so far. It is often necessary to group together tuples from one or more data sets, that are related in some way, so that they can

subsequently be processed together. This grouping operation is done by the COGROUP command.

Other Commands

Pig Latin has a number of other commands that are very similar to their SQL counterparts. These are:

1. UNION: Returns the union of two or more bags.
2. CROSS: Returns the cross product of two or more bags.
3. ORDER: Orders a bag by the specified field(s).
4. DISTINCT: Eliminates duplicate tuples in a bag. This command is just a shortcut for grouping the bag by all fields, and then projecting out the groups.

Future Work

Pig is a project under active development. New features are being added continually to improve the user experience and yield even higher productivity gains. There are a number of promising directions that are yet unexplored in the context of the Pig system.

- Safe Optimizer
- User Interfaces
- External Functions
- Unified Environment

Summary

Pig's target demographic is experienced procedural programmers who prefer map-reduce style programming over the more declarative, SQL-style programming, for stylistic reasons as well as the ability to control the execution plan. Pig aims for a sweet spot between these two extremes, offering high-level data manipulation primitives such as projection and join, but in a much less declarative style than SQL.

A novel debugging environment for Pig, called Pig Pen. It makes it easy and fast for users to construct and debug their programs in an incremental fashion. The Pig system compiles Pig Latin expressions into a sequence of map-reduce jobs, and orchestrates the execution of these jobs on Hadoop, an open-source scalable map-reduce implementation.

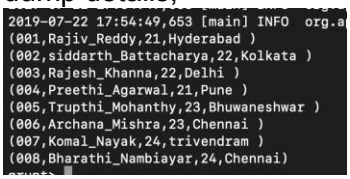
PART 5 - Programming Assignment

Execute one function of your choice from each group of commands, i.e., one function from Eval Functions, one function from Load/Store functions, one function from Math functions, etc, from the Official Pig website. Every time you execute a command copy-paste the screenshot, including the output, to a word document, and submit with your assignment.
http://pig.apache.org/docs/r0.17.0/func.html

- LOAD

```
2019-07-20 07:24:50,420 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabl
grunt> userData = LOAD '/pigData/users' USING PigStorage(':') AS (UserID,temp1,Gender,movieId,temp2,Genres);
grunt> store userData INTO '/pigData/output'
```

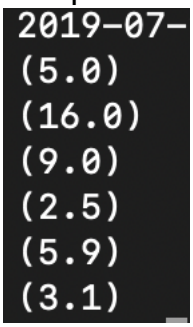
- TextLoader
  details = LOAD '/Assignment5/part5/textFile.txt' USING TextLoader();
  dump details;



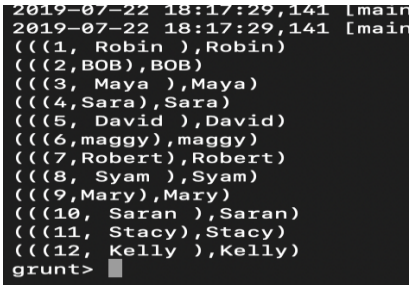
- **ABS**

```
5
16
9
2.5
5.9
3.1
```

  grunt> details = LOAD '/Assignment5/part5/absFile' USING PigStorage(',') as (data:float);
  dump details;



- TRIM
  empData = LOAD '/Users/ajaygoel/Documents/Neu/BigData/Assignments/Assignment5/trim' USING PigStorage(',') AS (id,name,age,location);
  trim_data = FOREACH empData GENERATE (id,name), TRIM(name);



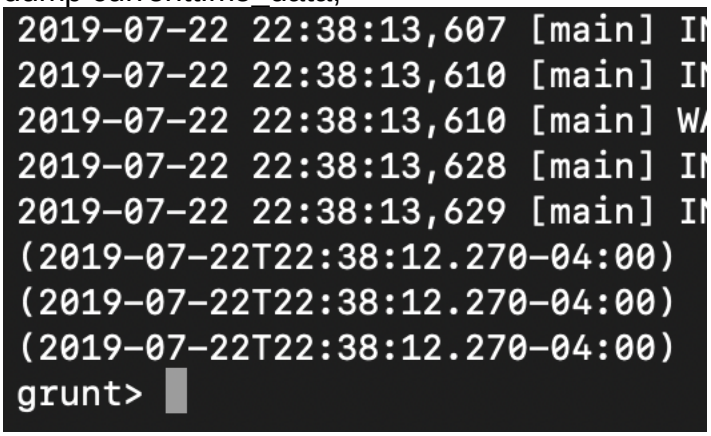
Dump trim_data;

- **CurrentTime();**

  date_data = LOAD '/Users/ajaygoel/Documents/Neu/BigData/Assignments/Assignment5/part5_date.txt' USING PigStorage(',') AS (id:int,date:chararray);

  currenttime_data = foreach date_data generate CurrentTime();

  dump currenttime_data;

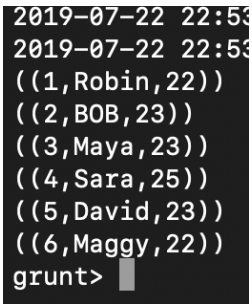


- **TOTUPLE():-**

  emp_data = LOAD '/Users/ajaygoel/Documents/Neu/BigData/Assignments/Assignment5/part5_employee.txt' USING PigStorage(',') AS (id:int, name:chararray, age:int, city:chararray);

  totuple = FOREACH emp_data GENERATE TOTUPLE (id,name,age);

  DUMP totuple;

- 

HIVE UDF
Done in part 6,7

PART 6 - Programming Assignment - Apache Pig (Use .pig scripts)
(1 million ratings from 6000 users on 4000 movies).
http://grouplens.org/datasets/movielens/

Task 1. Write a Pig Script to find the top 25 rated movies in the movieLens dataset
Task 2. Write a Pig Script to find the number of males and females in the movielens dataset
Task 3. Write a Pig Script to find the number of movies rated by different users

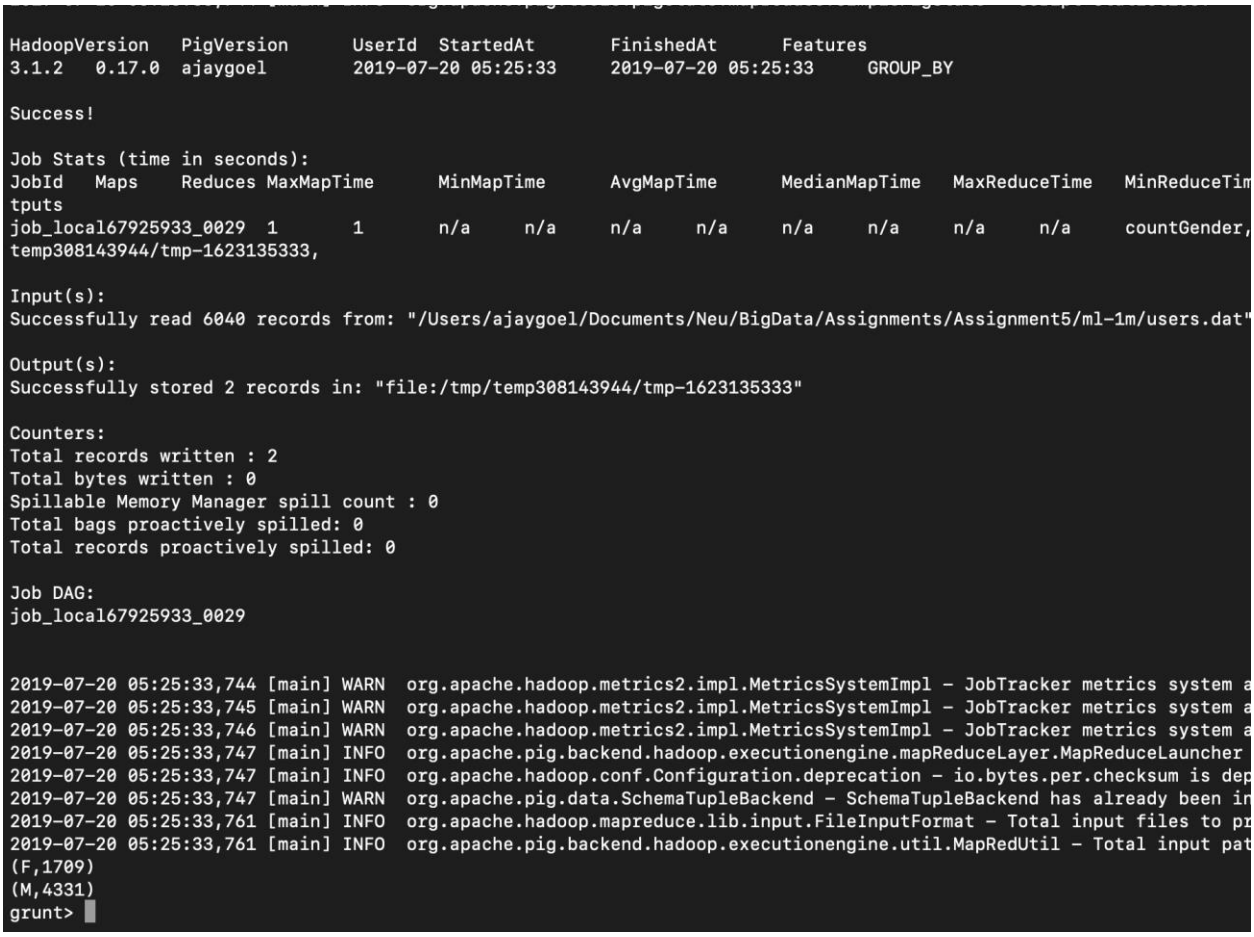
TASK 1:

exec /Users/ajaygoel/Documents/Neu/BigData/Assignments/Assignment5/part6_1.pig

```
2019-07-20 05:03:37,148 [main] INFO  org.apache.pig.backend.hadoop.executionengine.m
2019-07-20 05:03:37,149 [main] INFO  org.apache.hadoop.conf.Configuration.deprecatio
2019-07-20 05:03:37,149 [main] WARN  org.apache.pig.data.SchemaTupleBackend - Schema
2019-07-20 05:03:37,162 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInput
2019-07-20 05:03:37,162 [main] INFO  org.apache.pig.backend.hadoop.executionengine.u
(1,4.146846413095811,1,,Toy Story (1995),,Animation|Children's|Comedy)
(2,3.20114122681883,2,,Jumanji (1995),,Adventure|Children's|Fantasy)
(3,3.01673640167364,3,,Grumpier Old Men (1995),,Comedy|Romance)
(4,2.7294117647058824,4,,Waiting to Exhale (1995),,Comedy|Drama)
(5,3.0067567567567566,5,,Father of the Bride Part II (1995),,Comedy)
(6,3.8787234042553194,6,,Heat (1995),,Action|Crime|Thriller)
(7,3.410480349344978,7,,Sabrina (1995),,Comedy|Romance)
(8,3.014705882352941,8,,Tom and Huck (1995),,Adventure|Children's)
(9,2.656862745098039,9,,Sudden Death (1995),,Action)
(10,3.5405405405405403,10,,GoldenEye (1995),,Action|Adventure|Thriller)
(11,3.7938044530493706,11,,American President, The (1995),,Comedy|Drama|Romance)
(12,2.3625,12,,Dracula, Dead and Loving It (1995),)
(13,3.2626262626262625,13,,Balto (1995),,Animation|Children's)
(14,3.542483660130719,14,,Nixon (1995),,Drama)
(15,2.458904109589041,15,,Cutthroat Island (1995),,Action|Adventure|Romance)
(16,3.7932551319648096,16,,Casino (1995),,Drama|Thriller)
(17,4.027544910179641,17,,Sense and Sensibility (1995),,Drama|Romance)
(18,3.337579617834395,18,,Four Rooms (1995),,Thriller)
(19,2.480719794344473,19,,Ace Ventura, When Nature Calls (1995),)
(20,2.5375,20,,Money Train (1995),,Action)
(21,3.6238938053097347,21,,Get Shorty (1995),,Action|Comedy|Drama)
(22,3.3492063492063493,22,,Copycat (1995),,Crime|Drama|Thriller)
(23,2.857142857142857,23,,Assassins (1995),,Thriller)
(24,3.1794871794871793,24,,Powder (1995),,Drama|Sci-Fi)
(25,3.6510204081632653,25,,Leaving Las Vegas (1995),,Drama|Romance)
grunt>
```

Task2:

Task 2. Write a Pig Script to find the number of males and females in the movielens dataset



Task 3. Write a Pig Script to find the number of movies rated by different users

exec /Users/ajaygoel/Documents/Neu/BigData/Assignments/Assignment5/part6_3.pig

```
/usr/local/Cellar/hadoop/3.1.2/sbin — -bash ...    ....17.0/libexec/pig-0.17.0-core-h2.jar -x local    ...u/BigData/lab10/ml-latest-sm
Successfully stored 6040 records in: "file:/tmp/temp308143944/tmp1444436253"

Counters:
Total records written : 6040
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local749602947_0035


2019-07-20 05:51:33,101 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTrac
2019-07-20 05:51:33,102 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTrac
2019-07-20 05:51:33,103 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTrac
2019-07-20 05:51:33,104 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLaye
2019-07-20 05:51:33,104 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
2019-07-20 05:51:33,104 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
2019-07-20 05:51:33,119 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Tot
2019-07-20 05:51:33,119 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUt
(1,53)
(2,129)
(3,51)
(4,21)
(5,198)
(6,71)
(7,31)
(8,139)
(9,106)
(10,401)
(11,137)
(12,23)
(13,108)
(14,25)
(15,201)
(16,35)
(17,211)
(18,305)
(19,255)
(20,24)
(21,22)
(22,297)
(23,304)
(24,136)
(25,85)
(26,400)
(27,70)
(28,107)
(29,108)
(30,43)
```

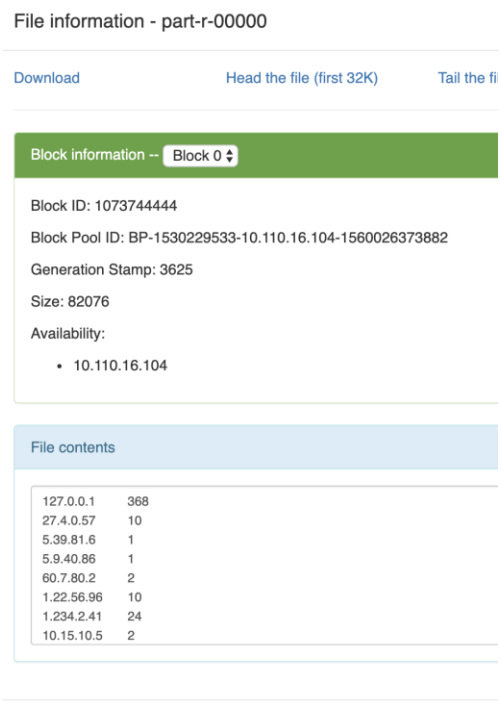
PART 7

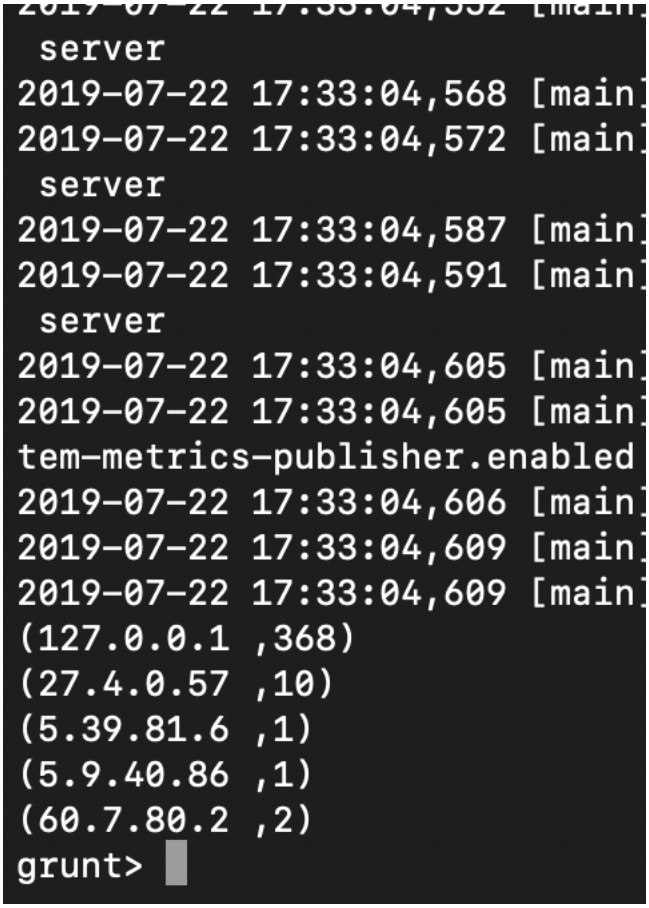
PART 7 - Programming Assignment - Apache Pig (Use GRUNT Shell)

Copy the ‘access.log’ file, used in previous assignments, into HDFS under /logs directory.
Using the access.log file stored in HDFS, implement Pig Script to find the number of times each IP accessed the website.

```
logsData = LOAD '/Assignment5/logs' USING PigStorage('-') AS (ipAddress,temp,data1,data2);
ipGroup = GROUP logsData BY ipAddress;
data = FOREACH ipGroup GENERATE group, COUNT(logsData.ipAddress);
STORE data INTO '/Assignment5/part7/logs'
```

Goel, Ajay: Nu Id (001897443)





Goel, Ajay: Nu Id (001897443)