Course: CS-5525-0001

Semester: Fall 2019

Project Report

ON

Big Data Benchmarks

Presented by:

Kranthi Kiran Reddy Vanga	-	16274077
Bhavaz Akula	-	16282638

MOTIVATION

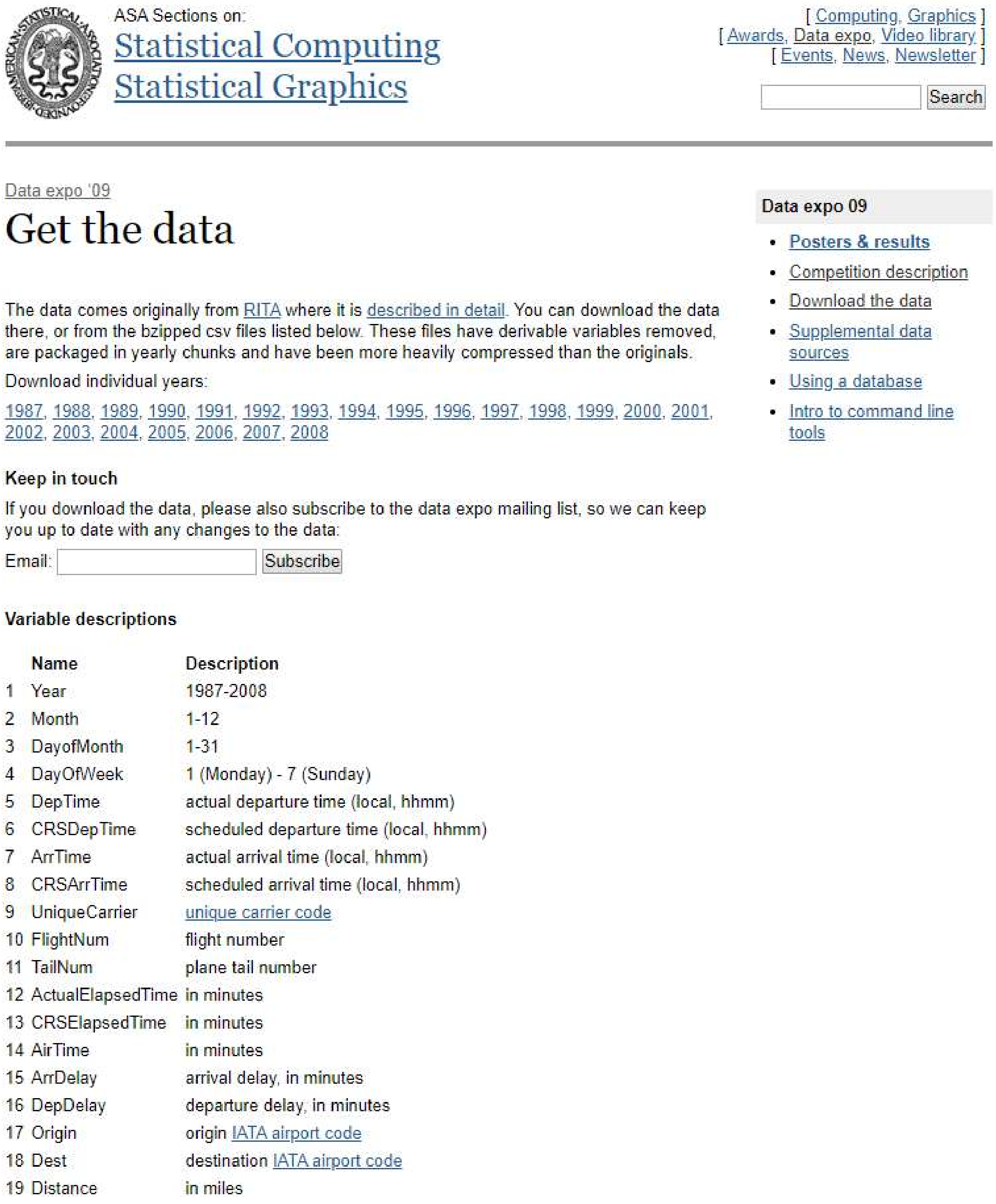
- Often we are overwhelmed with number of tools available for a purpose.
- Particularly, Big data ecosystem is exploding along with the data.
- Companies often tend to have most of them if not all and end up using very few.
- Buried under such a large landscape we want to find ways to breathe easy.

OBJECTIVE

- Big data benchmarks help user to find which tool has to be used for required purpose.
- We want to come up with a solution. 'Right tool at right time'.
- Have handful of tools useful for the purpose.
- We would use cloud services to set up Hadoop cluster and running environment for all the tools.
- Choose multiple objectives as a user and perform tasks using the tools Spark-SQL, Spark-Dataframe, Hive-Mapreduce, and Hive-Tez.
- Come up with numbers and factors the helps user to choose tool that suits his purpose.
- This saves user's time of setting up only relevant tools. Remind you setting up these tools is a lot involved process at scale which firms use.

DATASETS USED

- We have used the data set “Data expo 09” for this project.
- http://stat-computing.org/dataexpo/2009/the-data.html



Data expo '09

Get the data

The data comes originally from RITA where it is described in detail. You can download the data there, or from the bzipped csv files listed below. These files have derivable variables removed, are packaged in yearly chunks and have been more heavily compressed than the originals.

Download individual years:

1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008

Keep in touch

If you download the data, please also subscribe to the data expo mailing list, so we can keep you up to date with any changes to the data:

Email:

Variable descriptions

	Name	Description
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	unique carrier code
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes
13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes
17	Origin	origin IATA airport code
18	Dest	destination IATA airport code
19	Distance	in miles

Data expo '09

Using a database

For moderate initial investment in time, and a large investment in space (>30 gigabytes), you can considerably speed up access to the data by loading it into a database. This page shows you how to do so for sqlite, an open-source sql database.

Create database

Creating a new database couldn't be simpler: just run the following on the command line to create a new database in the current directory.

```
sqlite3 ontime.sqlite3
```

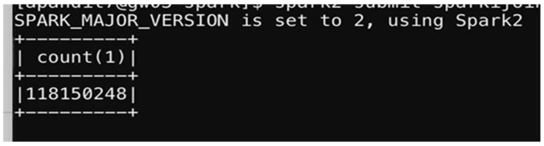
Create table and import data

Next create a table with fields that match the csv files:

```
create table ontime (
  Year int,
  Month int,
  DayofMonth int,
  DayOfWeek int,
  DepTime  int,
  CRSDepTime int,
  ArrTime int,
  CRSArrTime int,
  UniqueCarrier varchar(5),
  FlightNum int,
  TailNum varchar(8),
  ActualElapsedTime int,
  CRSElapsedTime int,
  AirTime int,
  ArrDelay int,
  DepDelay int,
  Origin varchar(3),
  Dest varchar(3),
  Distance int,
  TaxiIn int,
  TaxiOut int,
  Cancelled int,
  CancellationCode varchar(1),
  Diverted varchar(1),
  CarrierDelay int,
  WeatherDelay int,
  NASDelay int
```

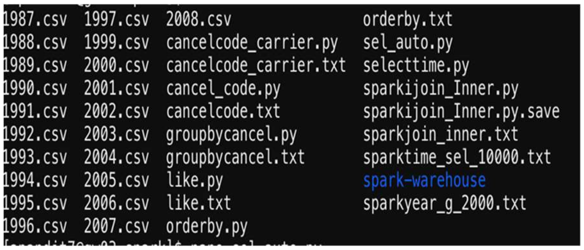
SIZE/YEAR DURATION

- The data used has a range from year 1987 to year 2019 of which consist of flight data having the details of year, month, airtime, etc.
- We have worked on the data set which consists of 120 millions of records.



```
[apandit7@gw03 spark]$ spark2-submit sparkijoin
SPARK_MAJOR_VERSION is set to 2, using Spark2
+---------+
| count(1)|
+---------+
|118150248|
+---------+
```

- Data engineering has been on this data and anomalies have been removed and the final data we used for Data analysis is around 34GB.

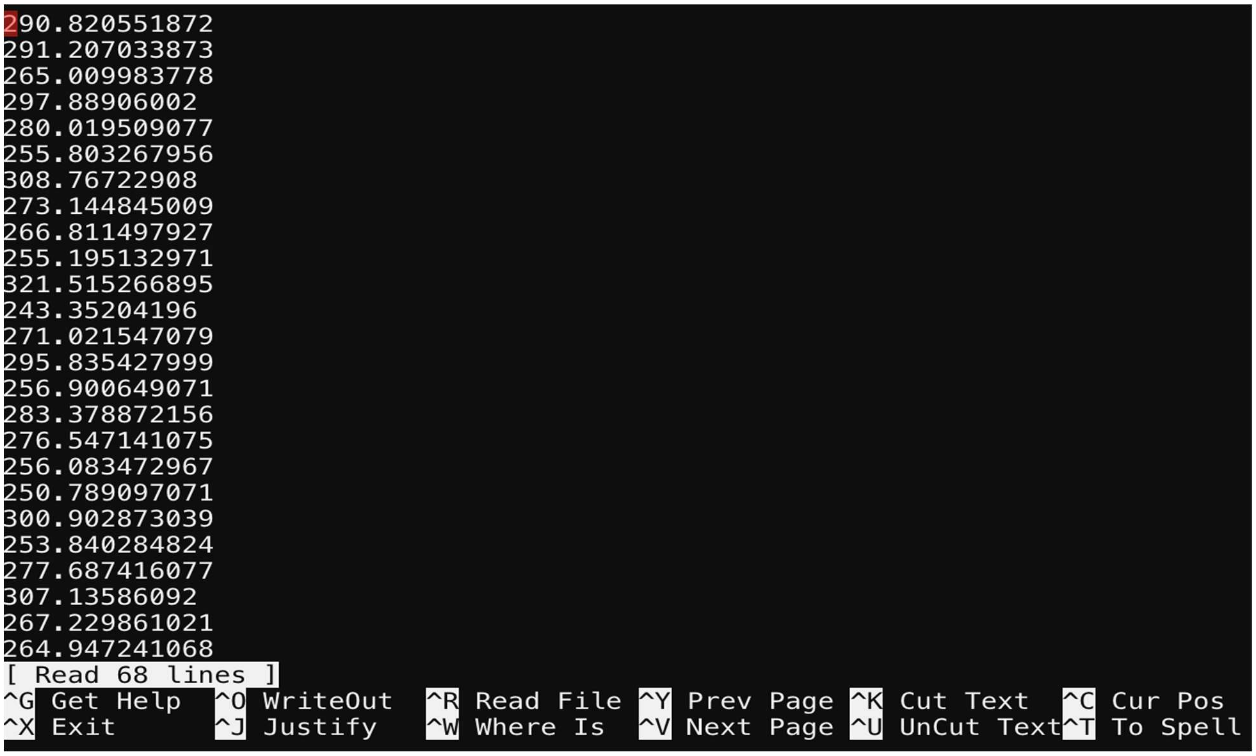


```
1987.csv  1997.csv  2008.csv                  orderby.txt
1988.csv  1999.csv  cancelcode_carrier.py     sel_auto.py
1989.csv  2000.csv  cancelcode_carrier.txt    selecttime.py
1990.csv  2001.csv  cancel_code.py            sparkijoin_Inner.py
1991.csv  2002.csv  cancelcode.txt            sparkijoin_Inner.py.save
1992.csv  2003.csv  groupbycancel.py          sparkjoin_inner.txt
1993.csv  2004.csv  groupbycancel.txt         sparktime_sel_10000.txt
1994.csv  2005.csv  like.py                   spark-warehouse
1995.csv  2006.csv  like.txt                  sparkyear_g_2000.txt
1996.csv  2007.csv  orderby.py
```

ANALYSIS DONE

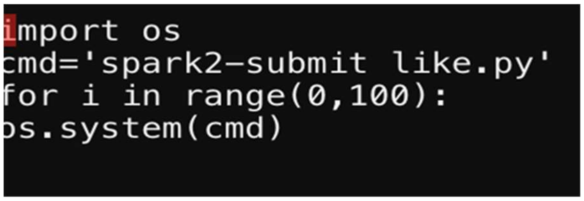
- Data has been refined and all the anomalies have been removed.
- Hadoop environment has been setup in ITversity Lab, which provides big data cluster with 400GB ram.
- Spark SQL, Spark DF, Hive MR and Hive on Tez frameworks have been setup on top of Hadoop.
- Data has been analysed and pushed into Hadoop cluster.
- To start with we have chosen querying tools that are popular.

- We executed the queries hundreds of time so that we can get accurate result.
- Performance evaluation has been on the Flight dataset.



```
290.820551872
291.207033873
265.009983778
297.88906002
280.019509077
255.803267956
308.76722908
273.144845009
266.811497927
255.195132971
321.515266895
243.35204196
271.021547079
295.835427999
256.900649071
283.378872156
276.547141075
256.083472967
250.789097071
300.902873039
253.840284824
277.687416077
307.13586092
267.229861021
264.947241068
[ Read 68 lines ]
^G Get Help    ^O WriteOut    ^R Read File ^Y Prev Page ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

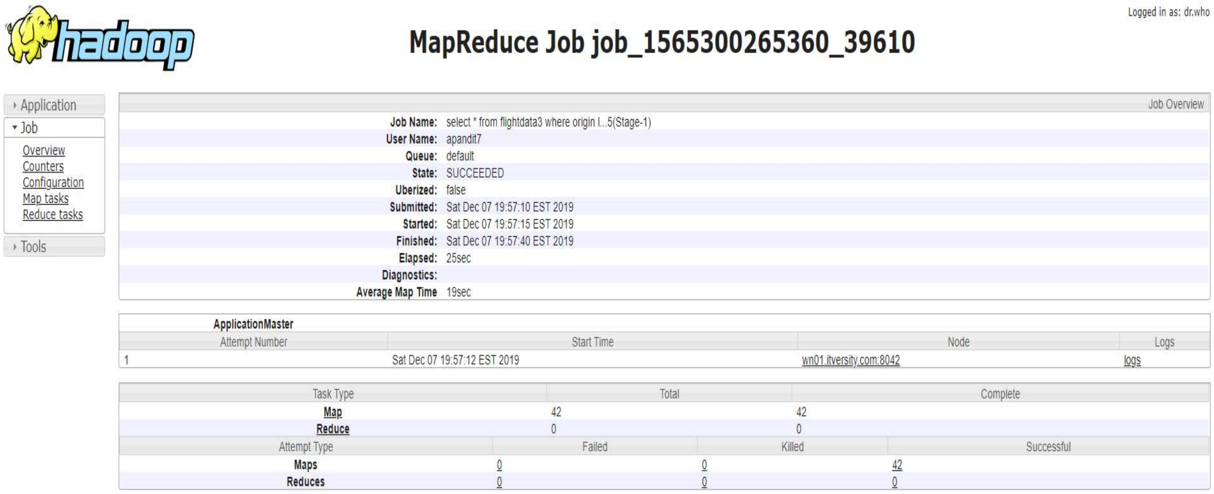
-
- We have run a python shell script to save the outputs.



```
import os
cmd='spark2-submit like.py'
for i in range(0,100):
os.system(cmd)
```

-
- The queries which we executed to determine the performance of different bigdata frameworks are count , group by, order by, etc

- We have taken all the Run times and took the average and compared with other tools.



MapReduce Job job_1565300265360_39610

Job Name:	select * from flightdata3 where origin l...5(Stage-1)
User Name:	apandit7
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Sat Dec 07 19:57:10 EST 2019
Started:	Sat Dec 07 19:57:15 EST 2019
Finished:	Sat Dec 07 19:57:40 EST 2019
Elapsed:	25sec
Diagnostics:	
Average Map Time	19sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Sat Dec 07 19:57:12 EST 2019	wn01.itversity.com:8042	logs

Task Type	Total	Complete
Map	42	42
Reduce	0	0

Attempt Type	Failed	Killed	Successful
Maps	0	0	42
Reduces	0	0	0

TOOLS AND TECHNOLOGIES USED

- ITversity Labs

- Big Data frameworks (Hive on Tez, Hive on Mapreduce, Spark Dataframe and Spark SQL)

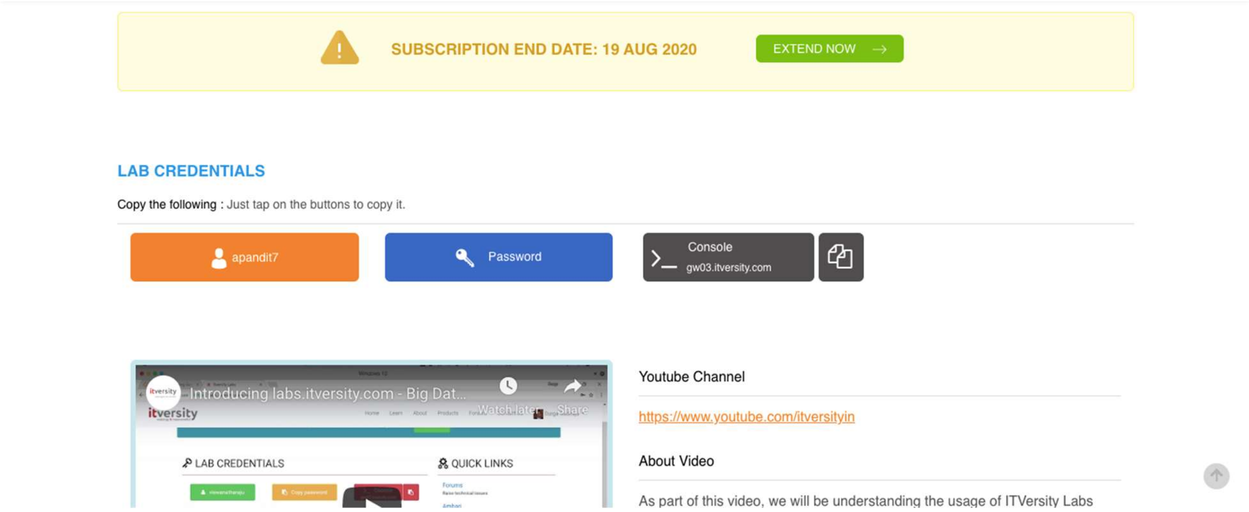
- Plotly

- Dash by Plotly

- Heroku Cloud

Execution

ITversity Home page



SUBSCRIPTION END DATE: 19 AUG 2020

LAB CREDENTIALS

Copy the following : Just tap on the buttons to copy it.

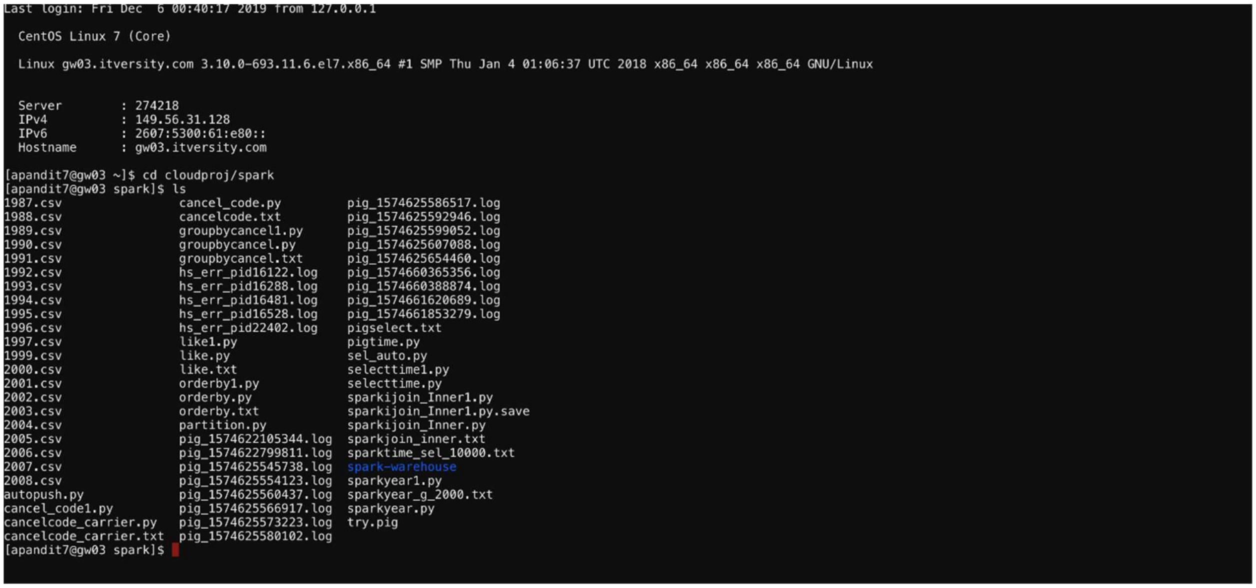
Youtube Channel

https://www.youtube.com/itversityin

About Video

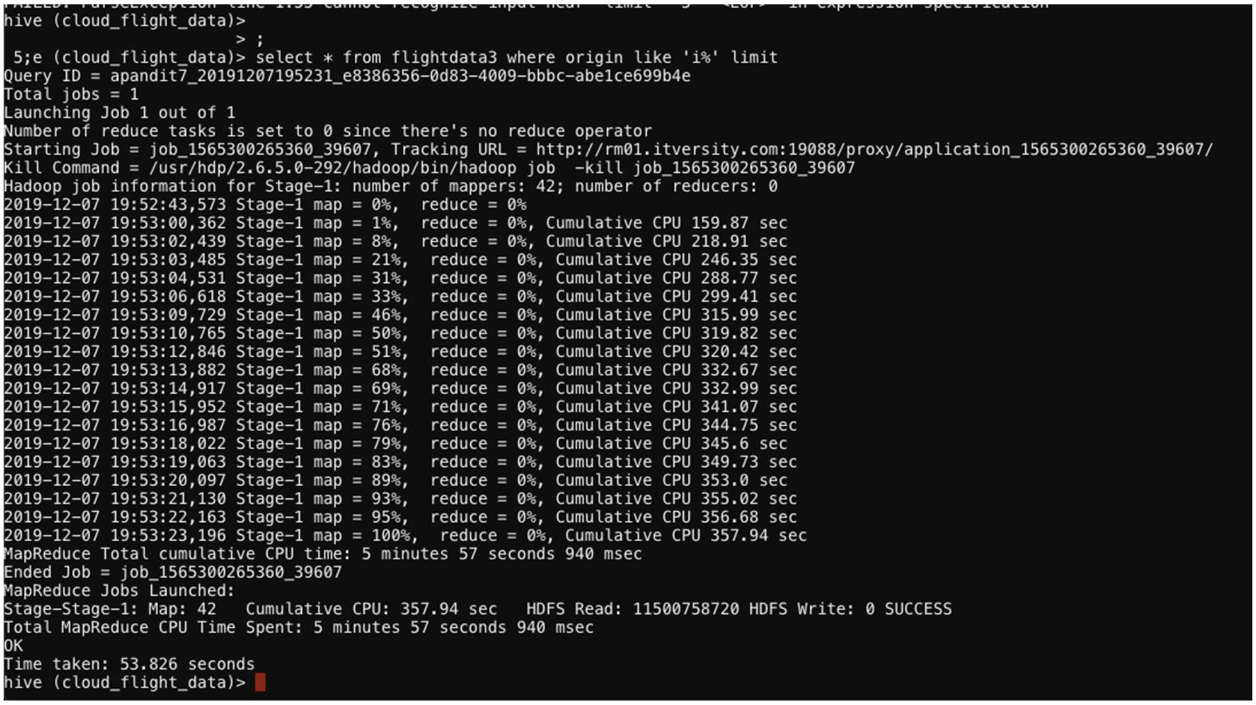
As part of this video, we will be understanding the usage of ITVersity Labs

Working Directory in Hadoop.



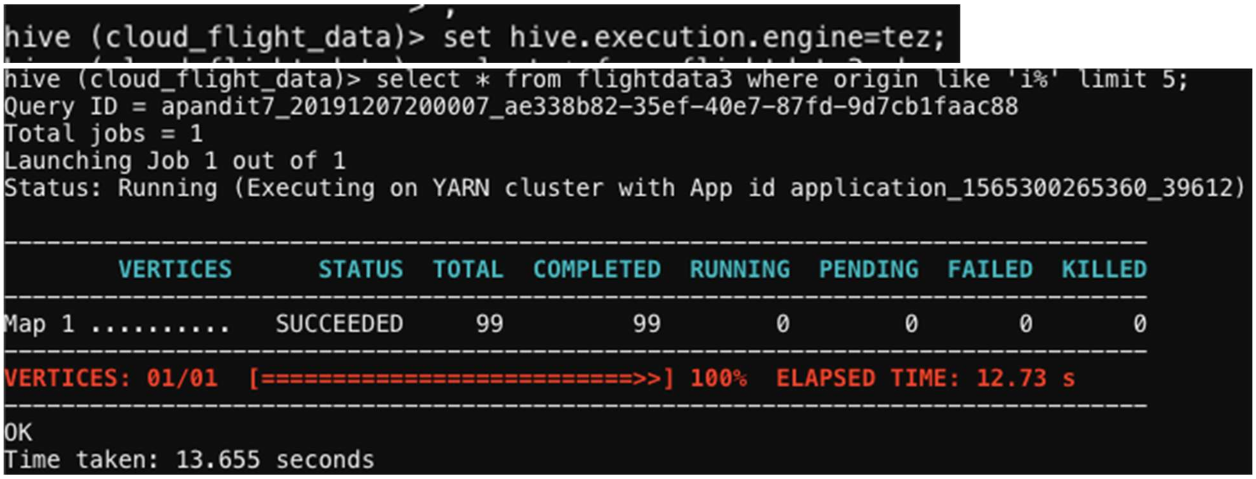
```
Last login: Fri Dec  6 00:40:17 2019 from 127.0.0.1

  CentOS Linux 7 (Core)

  Linux gw03.itversity.com 3.10.0-693.11.6.el7.x86_64 #1 SMP Thu Jan 4 01:06:37 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux


  Server        : 274218
  IPv4          : 149.56.31.128
  IPv6          : 2607:5300:61:e80::
  Hostname      : gw03.itversity.com

[apandit7@gw03 ~]$ cd cloudproj/spark
[apandit7@gw03 spark]$ ls
1987.csv                cancel_code.py         pig_1574625586517.log
1988.csv                cancelcode.txt         pig_1574625592946.log
1989.csv                groupbycancel1.py      pig_1574625599052.log
1990.csv                groupbycancel.py       pig_1574625607088.log
1991.csv                groupbycancel.txt      pig_1574625654460.log
1992.csv                hs_err_pid16122.log    pig_1574660365356.log
1993.csv                hs_err_pid16288.log    pig_1574660388874.log
1994.csv                hs_err_pid16481.log    pig_1574661620689.log
1995.csv                hs_err_pid16528.log    pig_1574661853279.log
1996.csv                hs_err_pid22402.log    pigselect.txt
1997.csv                like1.py               pigtime.py
1999.csv                like.py                sel_auto.py
2000.csv                like.txt               selecttime1.py
2001.csv                orderby1.py            selecttime.py
2002.csv                orderby.py             sparkijoin_Inner1.py
2003.csv                orderby.txt            sparkijoin_Inner1.py.save
2004.csv                partition.py           sparkijoin_Inner.py
2005.csv                pig_1574622105344.log  sparkjoin_inner.txt
2006.csv                pig_1574622799811.log  sparktime_sel_10000.txt
2007.csv                pig_1574625545738.log  spark-warehouse
2008.csv                pig_1574625554123.log  sparkyear1.py
autopush.py             pig_1574625560437.log  sparkyear_g_2000.txt
cancel_code1.py         pig_1574625566917.log  sparkyear.py
cancelcode_carrier.py   pig_1574625573223.log  try.pig
cancelcode_carrier.txt  pig_1574625580102.log
[apandit7@gw03 spark]$ 
```

One of the Query execution in Hive MapReduce framework.



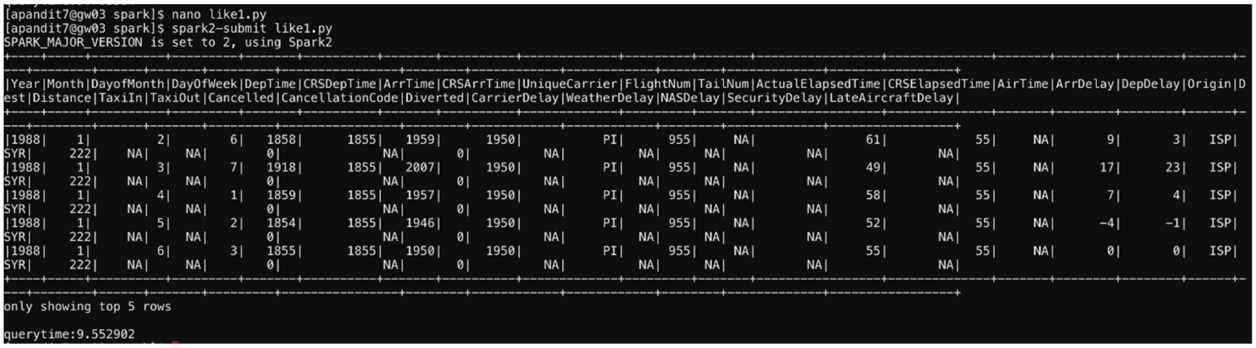
```
FAILED: ParseException line 1:33 cannot recognize input near 'limit' '5' '<EOF>' in expression specification
hive (cloud_flight_data)>
                        > ;
 5;e (cloud_flight_data)> select * from flightdata3 where origin like 'i%' limit
Query ID = apandit7_20191207195231_e8386356-0d83-4009-bbbc-abe1ce699b4e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1565300265360_39607, Tracking URL = http://rm01.itversity.com:19088/proxy/application_1565300265360_39607/
Kill Command = /usr/hdp/2.6.5.0-292/hadoop/bin/hadoop job  -kill job_1565300265360_39607
Hadoop job information for Stage-1: number of mappers: 42; number of reducers: 0
2019-12-07 19:52:43,573 Stage-1 map = 0%,  reduce = 0%
2019-12-07 19:53:00,362 Stage-1 map = 1%,  reduce = 0%, Cumulative CPU 159.87 sec
2019-12-07 19:53:02,439 Stage-1 map = 8%,  reduce = 0%, Cumulative CPU 218.91 sec
2019-12-07 19:53:03,485 Stage-1 map = 21%,  reduce = 0%, Cumulative CPU 246.35 sec
2019-12-07 19:53:04,531 Stage-1 map = 31%,  reduce = 0%, Cumulative CPU 288.77 sec
2019-12-07 19:53:06,618 Stage-1 map = 33%,  reduce = 0%, Cumulative CPU 299.41 sec
2019-12-07 19:53:09,729 Stage-1 map = 46%,  reduce = 0%, Cumulative CPU 315.99 sec
2019-12-07 19:53:10,765 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 319.82 sec
2019-12-07 19:53:12,846 Stage-1 map = 51%,  reduce = 0%, Cumulative CPU 320.42 sec
2019-12-07 19:53:13,882 Stage-1 map = 68%,  reduce = 0%, Cumulative CPU 332.67 sec
2019-12-07 19:53:14,917 Stage-1 map = 69%,  reduce = 0%, Cumulative CPU 332.99 sec
2019-12-07 19:53:15,952 Stage-1 map = 71%,  reduce = 0%, Cumulative CPU 341.07 sec
2019-12-07 19:53:16,987 Stage-1 map = 76%,  reduce = 0%, Cumulative CPU 344.75 sec
2019-12-07 19:53:18,022 Stage-1 map = 79%,  reduce = 0%, Cumulative CPU 345.6 sec
2019-12-07 19:53:19,063 Stage-1 map = 83%,  reduce = 0%, Cumulative CPU 349.73 sec
2019-12-07 19:53:20,097 Stage-1 map = 89%,  reduce = 0%, Cumulative CPU 353.0 sec
2019-12-07 19:53:21,130 Stage-1 map = 93%,  reduce = 0%, Cumulative CPU 355.02 sec
2019-12-07 19:53:22,163 Stage-1 map = 95%,  reduce = 0%, Cumulative CPU 356.68 sec
2019-12-07 19:53:23,196 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 357.94 sec
MapReduce Total cumulative CPU time: 5 minutes 57 seconds 940 msec
Ended Job = job_1565300265360_39607
MapReduce Jobs Launched:
Stage-Stage-1: Map: 42   Cumulative CPU: 357.94 sec   HDFS Read: 11500758720 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 5 minutes 57 seconds 940 msec
OK
Time taken: 53.826 seconds
hive (cloud_flight_data)>
```

One of the Query execution in Hive on Tez



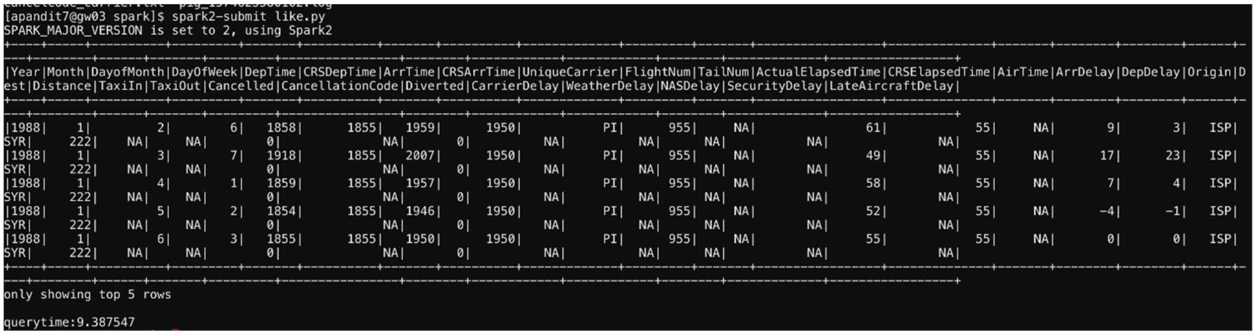
```
                        > ;
hive (cloud_flight_data)> set hive.execution.engine=tez;
hive (cloud_flight_data)> select * from flightdata3 where origin like 'i%' limit 5;
Query ID = apandit7_20191207200007_ae338b82-35ef-40e7-87fd-9d7cb1faac88
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1565300265360_39612)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED     99         99        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 12.73 s
----------------------------------------------------------------------------------------------
OK
Time taken: 13.655 seconds
```

One of the Query execution in Spark- Dataframe



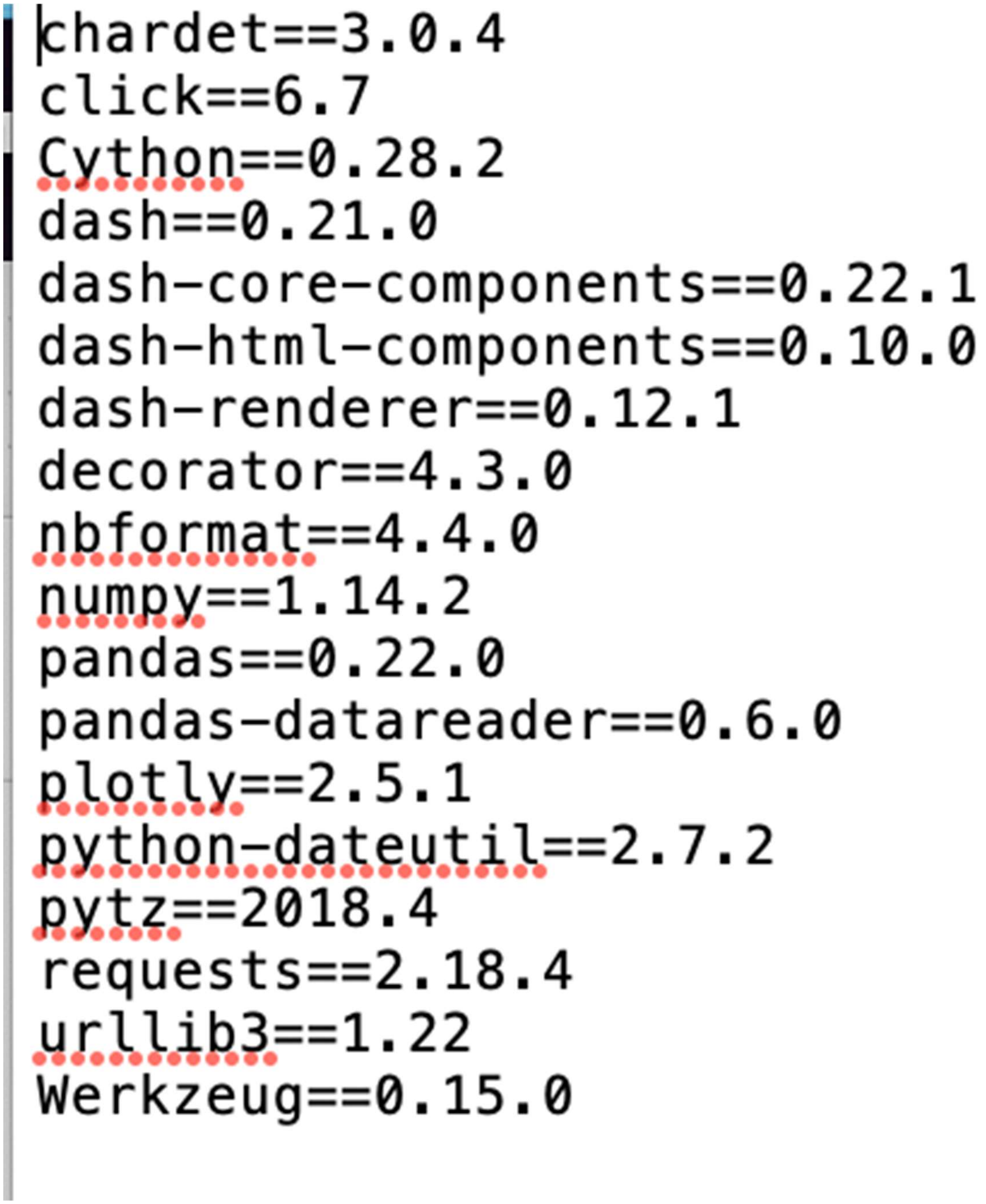
```
[apandit7@gw03 spark]$ nano like1.py
[apandit7@gw03 spark]$ spark2-submit like1.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-------+-----------------+--------------+-------+--------+--------+------+-
---+--------+------+-------+---------+----------------+--------+------------+------------+--------+-------------+-----------------+
|Year|Month|DayofMonth|DayOfWeek|DepTime|CRSDepTime|ArrTime|CRSArrTime|UniqueCarrier|FlightNum|TailNum|ActualElapsedTime|CRSElapsedTime|AirTime|ArrDelay|DepDelay|Origin|D
est|Distance|TaxiIn|TaxiOut|Cancelled|CancellationCode|Diverted|CarrierDelay|WeatherDelay|NASDelay|SecurityDelay|LateAircraftDelay|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-------+-----------------+--------------+-------+--------+--------+------+-
---+--------+------+-------+---------+----------------+--------+------------+------------+--------+-------------+-----------------+
|1988|    1|         2|        6|   1858|      1855|   1959|      1950|           PI|      955|     NA|               61|            55|     NA|       9|       3|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         3|        7|   1918|      1855|   2007|      1950|           PI|      955|     NA|               49|            55|     NA|      17|      23|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         4|        1|   1859|      1855|   1957|      1950|           PI|      955|     NA|               58|            55|     NA|       7|       4|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         5|        2|   1854|      1855|   1946|      1950|           PI|      955|     NA|               52|            55|     NA|      -4|      -1|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         6|        3|   1855|      1855|   1950|      1950|           PI|      955|     NA|               55|            55|     NA|       0|       0|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-------+-----------------+--------------+-------+--------+--------+------+-
---+--------+------+-------+---------+----------------+--------+------------+------------+--------+-------------+-----------------+
only showing top 5 rows

querytime:9.552902
```

One of the Query execution in Spark-SQL



```
[apandit7@gw03 spark]$ spark2-submit like.py
SPARK_MAJOR_VERSION is set to 2, using Spark2
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-------+-----------------+--------------+-------+--------+--------+------+-
---+--------+------+-------+---------+----------------+--------+------------+------------+--------+-------------+-----------------+
|Year|Month|DayofMonth|DayOfWeek|DepTime|CRSDepTime|ArrTime|CRSArrTime|UniqueCarrier|FlightNum|TailNum|ActualElapsedTime|CRSElapsedTime|AirTime|ArrDelay|DepDelay|Origin|D
est|Distance|TaxiIn|TaxiOut|Cancelled|CancellationCode|Diverted|CarrierDelay|WeatherDelay|NASDelay|SecurityDelay|LateAircraftDelay|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-------+-----------------+--------------+-------+--------+--------+------+-
---+--------+------+-------+---------+----------------+--------+------------+------------+--------+-------------+-----------------+
|1988|    1|         2|        6|   1858|      1855|   1959|      1950|           PI|      955|     NA|               61|            55|     NA|       9|       3|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         3|        7|   1918|      1855|   2007|      1950|           PI|      955|     NA|               49|            55|     NA|      17|      23|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         4|        1|   1859|      1855|   1957|      1950|           PI|      955|     NA|               58|            55|     NA|       7|       4|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         5|        2|   1854|      1855|   1946|      1950|           PI|      955|     NA|               52|            55|     NA|      -4|      -1|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
|1988|    1|         6|        3|   1855|      1855|   1950|      1950|           PI|      955|     NA|               55|            55|     NA|       0|       0|   ISP|
SYR|     222|    NA|     NA|        0|              NA|       0|          NA|          NA|      NA|           NA|               NA|
+----+-----+----------+---------+-------+----------+-------+----------+-------------+---------+-------+-----------------+--------------+-------+--------+--------+------+-
---+--------+------+-------+---------+----------------+--------+------------+------------+--------+-------------+-----------------+
only showing top 5 rows

querytime:9.387547
```

Requirements.txt for Plotly-Dash

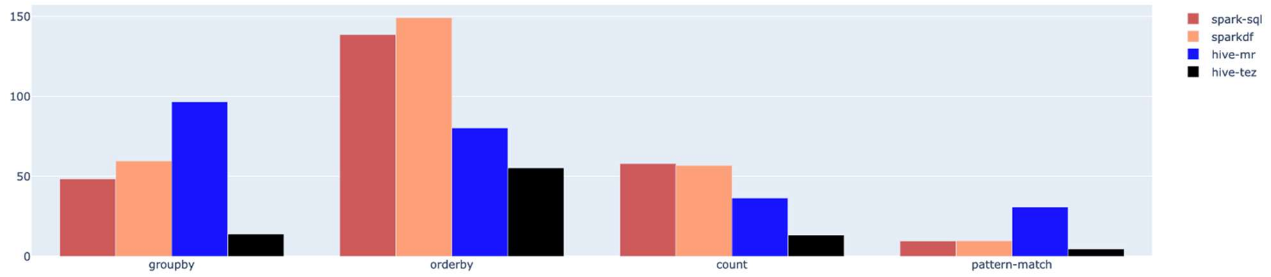


```
chardet==3.0.4
click==6.7
Cython==0.28.2
dash==0.21.0
dash-core-components==0.22.1
dash-html-components==0.10.0
dash-renderer==0.12.1
decorator==4.3.0
nbformat==4.4.0
numpy==1.14.2
pandas==0.22.0
pandas-datareader==0.6.0
plotly==2.5.1
python-dateutil==2.7.2
pytz==2018.4
requests==2.18.4
urllib3==1.22
Werkzeug==0.15.0
```

We have done entire project in python server rather than using the routine HTML/CSS or Angular frameworks with is one of the important highlights of this project.

Work Accomplished

Here is the comparision of 4 famous queries:

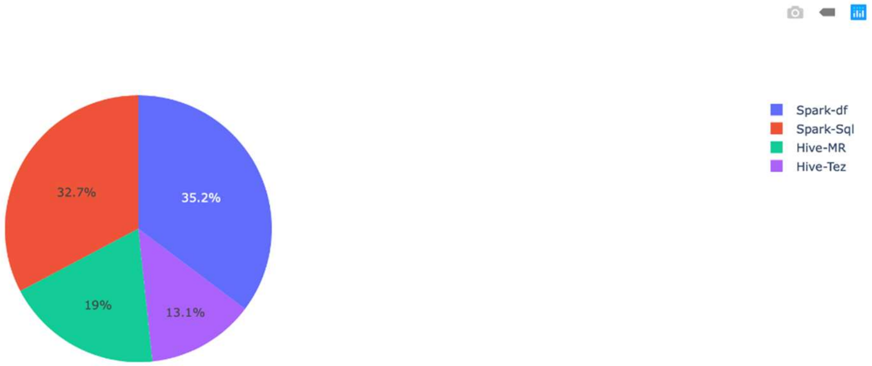


`*The data used here is of size 34GB and queries are run multiple times before avereging the time required`

Results:

1. For group by it can be seen that spark performs better than hive-Mapreduce and spark-SQL and spark-df results are comparable, Tez has overall upper hand.
2. This is because spark does in memory computation, but this does not mean spark is always faster.
3. For queries like ordering, counting where indexing is required spark performs poorly when compared to hive and hive on Tez performs better than hive.
4. So, though it can be said that spark is faster general here are few cases where hive on Tez performs better in general.
5. We can see that in case of pattern matching spark and Tez performs better and Tez is slightly better than spark, also, spark-SQL and spark-df perform equally.
6. So, Tez which enhances Mapreduce by DAG graphs performs reliably in all cases.

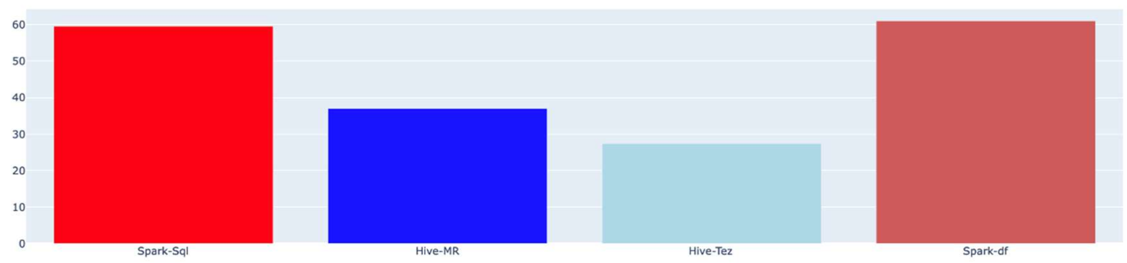
Counting Number of rows in the Dataset:



1. We can see that in case of counting which requires seraching through whole dataset spark performs poorly as compared to hive as spark has no Indexing functionality
2. Among hive on mapreduce and hive on tez frameworks Tez performs better as it is there to enhance MR functionality
3. Among spark API's spark-df and spark-sql perform equivqlently.

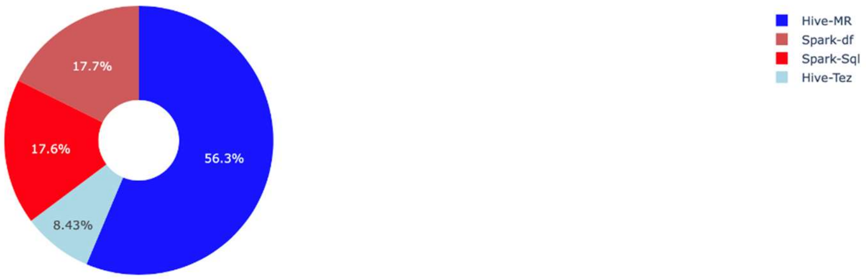
```
*Remember that we are counting almost 120 million rows
```

Counting Number of rows With a where Clause:



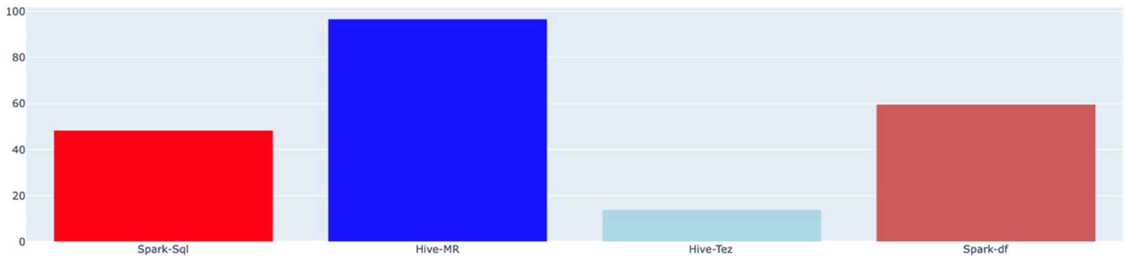
1. In this case Tez performs the best among all others
2. As counting is in volved in addition to where clause pespark performs poorly as compared to hive
3. When spark-sql and spark-df are compared both are equivalent and there is not much to seperate.

Pattern matching for origin starting with I:



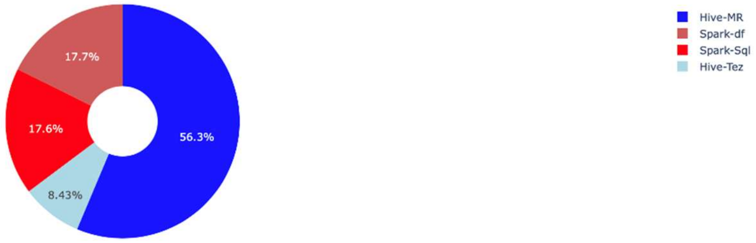
1. In this case tez is better than spark
2. Hive on mapreduce is the poorest of all
3. Spark-sql and spark-df both performed equivalently

Group By Query:



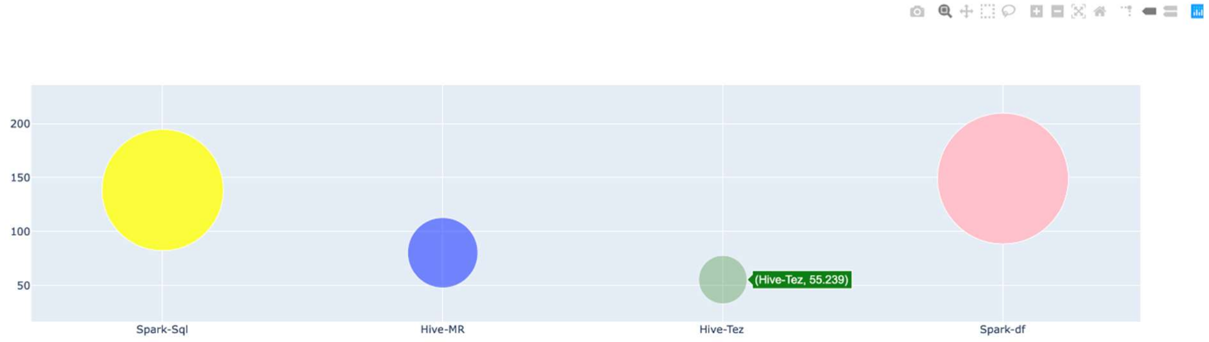
1. Again Tez was better in this case because of DAG and MR optimization.
2. Among spark-sql and spark-df spark-sql performed better as groupby is aggregation intensive task its generally rule that spark-sql performs better.
3. Hive-MR is poor option as there is lot of latency.

Finding for flights cancelled with weather delay:



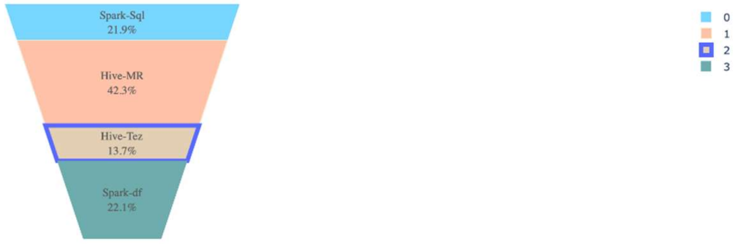
1. Here tez out performs all the other counter parts as it.
2. As it is simple search on large data Spark performs better than Hive-mapreduce.
3. Again performance of Spark-sql and spark Dataframes is equivalent as it does does not have any complex aggregation task.
4. As a general rule when involving simple searches avoid Hive-MR

Ordering Data:



1. We can see that yello and pink dots being less transparent corresponds to larger latency so spark is poor in ordering as it requires indexing
2. Tez usually performs better as it uses partial indexing of columns and enhance MR ------> DAG.
3. Hive MR also out performs spart due to non-existant indexing capability of spark.

Grouping and Ordering:



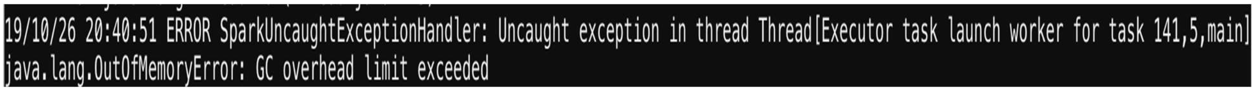
1. Tez outperforms spark and hive MR it is twice as fast as spark and 4 times faster than hive-MR.
2. Spark-sql and spark-df performed equivalently as shown in the figure they share equal area.
3. Hive MR performs poorly as it is computationally intensive to group and order.
4. We can conclude that tez and spark are reasonable choices in this case.

Conclusions:

1. Though both spark and Tez claims 100x speed than MR for in memory computation, In real time it is in the order of 10x or even much lesser.
2. Spark execution engine lacks the power of indexing on structured data and so performs poorly when we are trying to order the data or when we are trying to count.
3. Hive use Hadoop as its storage engine and runs only on HDFS.
4. Because of support for ANSI SQL standards, Hive can be integrated with databases like HBase and Cassandra, these tools has limited support for SQL and can help applications perform analytics and report on large data sets. Hive can also be integrated with data streaming tools such as Spark, Kafka and Flume.
5. Hive is pure data warehousing database which stores data in the form of tables. As a result, it can only process structured data read and written using SQL queries. Hive is not an option for unstructured data. In addition hive is not ideal for OLTP or OLAP kinds of operations.
6. Hive has partial indexing capabilities which helps when compared to spark while sorting and counting data.
7. The core strength of Spark is its ability to perform complex in-memory analytics and stream data sizing up to peta-bytes, making it more efficient and faster than map-reduce.
8. It can work on semi structured data and has streaming capabilities, though spark has many advantages it fails in the case of sorting and counting when compared to its counter parts.
9. Tez generalizes the map reduce paradigm by treating computations as DAGs. MapReduce tasks are combined into single job that is treated as node in the DAGs, thus enforcing concurrency and serialization.
10. Because of its core idea Tez performs better in most of the scenarios as it includes advantages of DAGs in spark and capabilities of map reduce.
11. In case of performing Joins at peta-byte scale it is general rule to use Hive on Tez rather than spark

ISSUES FACED

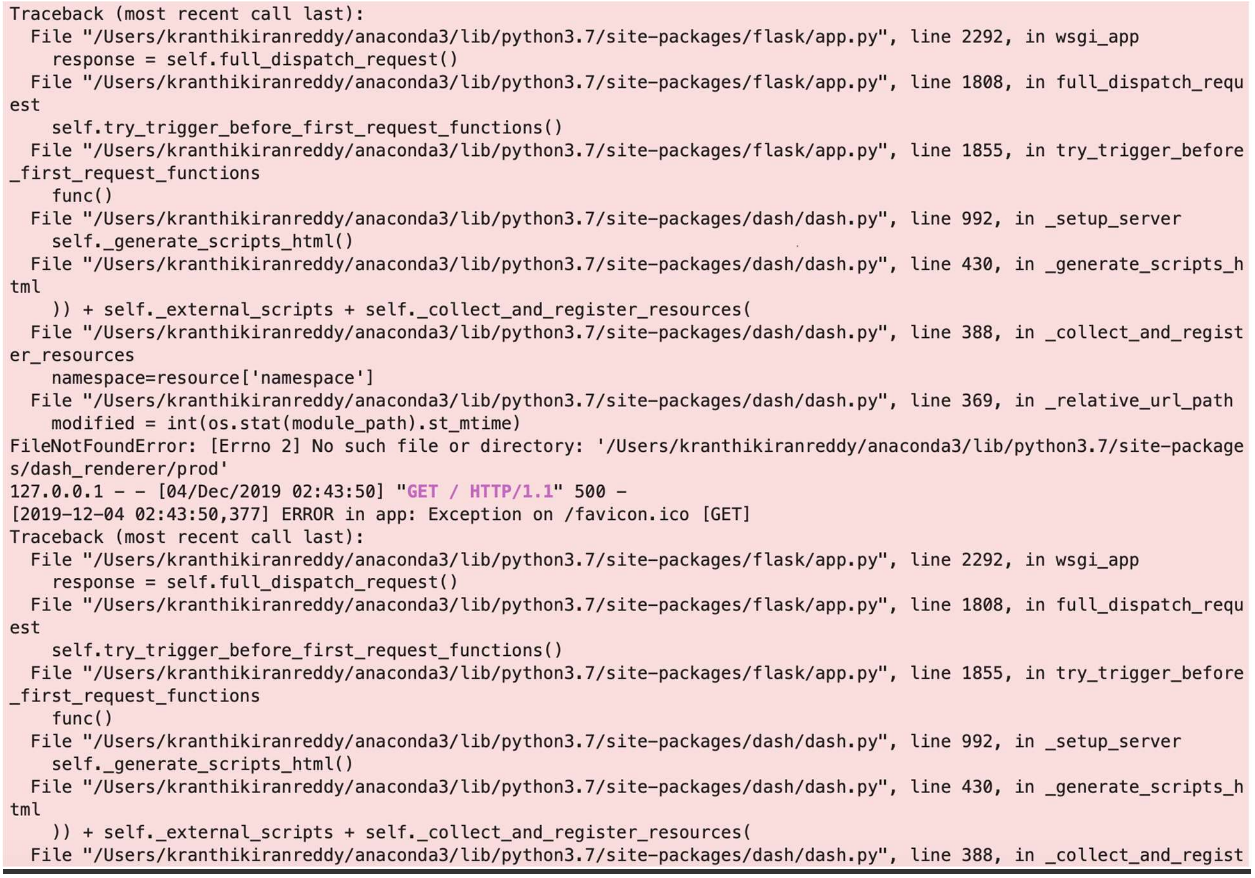
- GC overhead error for running out of memory.



```
19/10/26 20:40:51 ERROR SparkUncaughtExceptionHandler: Uncaught exception in thread Thread[Executor task launch worker for task 141,5,main]
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

This error was due to the Cartesian product nature of join operation which unfortunately could not be overcome so we scaled the data down for join operation. Query Times for join is more than 300 seconds with data scaled down to 1000 times

  - As part of our project objective we had to run each query
    - 100 times
- Data engineering at such large scale was notoriously time taking.
- We have compatibility issues in Plotly



```
Traceback (most recent call last):
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/flask/app.py", line 2292, in wsgi_app
    response = self.full_dispatch_request()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/flask/app.py", line 1808, in full_dispatch_requ
est
    self.try_trigger_before_first_request_functions()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/flask/app.py", line 1855, in try_trigger_before
_first_request_functions
    func()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 992, in _setup_server
    self._generate_scripts_html()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 430, in _generate_scripts_h
tml
    )) + self._external_scripts + self._collect_and_register_resources(
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 388, in _collect_and_regist
er_resources
    namespace=resource['namespace']
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 369, in _relative_url_path
    modified = int(os.stat(module_path).st_mtime)
FileNotFoundError: [Errno 2] No such file or directory: '/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-package
s/dash_renderer/prod'
127.0.0.1 - - [04/Dec/2019 02:43:50] "GET / HTTP/1.1" 500 -
[2019-12-04 02:43:50,377] ERROR in app: Exception on /favicon.ico [GET]
Traceback (most recent call last):
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/flask/app.py", line 2292, in wsgi_app
    response = self.full_dispatch_request()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/flask/app.py", line 1808, in full_dispatch_requ
est
    self.try_trigger_before_first_request_functions()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/flask/app.py", line 1855, in try_trigger_before
_first_request_functions
    func()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 992, in _setup_server
    self._generate_scripts_html()
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 430, in _generate_scripts_h
tml
    )) + self._external_scripts + self._collect_and_register_resources(
  File "/Users/kranthikiranreddy/anaconda3/lib/python3.7/site-packages/dash/dash.py", line 388, in _collect_and_regist
```

LEARNING OUTCOMES

- Working on this project is a knowledgeable process as we got to learn on how to work with Big Data Tools.
- Data Analysis
- Storing and deploying on Cloud Servers.
- New application development architecture
- Knowledge on new frameworks, Spark, Hive, Tez, Hadoop, and Plotly-Dash has been achieved.
- This project is a collaborative success.

REFERENCES

- Compared our project with Hadoop DB Paper :http://www.cs.umd.edu/~abadi/papers/hadoopdb.pdf
- Expo Data 09 : http://stat-computing.org/dataexpo/2009/the-data.html
- Plotly : https://plot.ly/python/
- ITversity : http://discuss.itversity.com/categories

LINKS

- **Project link:** http://kkrv.herokuapp.com/

- **GitHub link:** https://github.com/Bhavaz/2019-Fall-CC-Team6-Big-Data-Benchmarks