# BDA
# Assignment 2

Aakash Tanwar 2016215

Priya Rajpurohit 2015073

# Part 1: Postgres with Apache Spark

# How many records does the table contain?

```
scala> val query="(select count(*) from ghtorrent) as query"
query: String = (select count(*) from ghtorrent) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]

scala> personDf.show()
+-------+
|  count|
+-------+
|9669634|
+-------+
```

# Count the number of WARNing messages.

```
scala> val query="(SELECT COUNT(*) FROM ghtorrent WHERE log_level='WARN') as query"
query: String = (SELECT COUNT(*) FROM ghtorrent WHERE log_level='WARN') as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]

scala> personDf.show()
+------+
| count|
+------+
|132158|
+------+
```

# How many repositories were processed in total?

```
scala> val query="(SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE repo <>'' AND information LIKE '%URL%repos%/%?%' AND ret_stage='api_client.rb') as query"
query: String = (SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE repo <>'' AND information LIKE '%URL%repos%/%?%' AND ret_stage='api_client.rb') as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]

scala> personDf.show()
+------+
| count|
+------+
|381194|
+------+
```

# Which 10 clients did the highest HTTP requests?

```
scala> val query="(SELECT downloader_id,COUNT(*) FROM ghtorrent WHERE information LIKE '%URL: https%' GROUP BY downloader_id ORDER BY COUNT(*) DESC LIMIT 10) as query"
query: String = (SELECT downloader_id,COUNT(*) FROM ghtorrent WHERE information LIKE '%URL: https%' GROUP BY downloader_id ORDER BY COUNT(*) DESC LIMIT 10) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [downloader_id: string, count: bigint]

scala> personDf.show()
+-------------+-----+
|downloader_id|count|
+-------------+-----+
| ghtorrent-13|85528|
|  ghtorrent-4|19046|
| ghtorrent-18|18948|
| ghtorrent-10|18926|
| ghtorrent-40|18911|
| ghtorrent-39|18616|
| ghtorrent-38|18614|
| ghtorrent-47|18604|
|  ghtorrent-1|18463|
| ghtorrent-24|18452|
+-------------+-----+
```

# Which 10 client did the highest FAILED HTTP requests? (Operation part starts with the string ”Failed”)

```
scala> val query="(SELECT downloader_id,COUNT(*) FROM ghtorrent WHERE information LIKE '%Failed%URL: https%' GROUP BY downloader_id ORDER BY COUNT(*) DESC LIMIT 10) as query"
query: String = (SELECT downloader_id,COUNT(*) FROM ghtorrent WHERE information LIKE '%Failed%URL: https%' GROUP BY downloader_id ORDER BY COUNT(*) DESC LIMIT 10) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [downloader_id: string, count: bigint]

scala> personDf.show()
+-------------+-----+
|downloader_id|count|
+-------------+-----+
| ghtorrent-13|79623|
| ghtorrent-21| 1378|
| ghtorrent-40| 1134|
| ghtorrent-18|  368|
| ghtorrent-42|  357|
|  ghtorrent-9|  356|
|  ghtorrent-4|  352|
| ghtorrent-25|  342|
| ghtorrent-22|  333|
|  ghtorrent-6|  332|
+-------------+-----+
```

# What is the most active hour of day?

```
scala> val query="(SELECT hour,COUNT(*) FROM ghtorrent GROUP BY hour ORDER BY COUNT(*) DESC LIMIT 1) as query"
query: String = (SELECT hour,COUNT(*) FROM ghtorrent GROUP BY hour ORDER BY COUNT(*) DESC LIMIT 1) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [hour: string, count: bigint]

scala> personDf.show()
+----+-------+
|hour|  count|
+----+-------+
|  10|2662487|
+----+-------+
```

# What is the most active repository?

```
scala> val query="(SELECT repo,COUNT(*) FROM ghtorrent WHERE repo<>'' GROUP BY repo ORDER BY COUNT(*) DESC LIMIT 1) as query"
query: String = (SELECT repo,COUNT(*) FROM ghtorrent WHERE repo<>'' GROUP BY repo ORDER BY COUNT(*) DESC LIMIT 1) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [repo: string, count: bigint]

scala> personDf.show()
+--------------------+-----+
|                repo|count|
+--------------------+-----+
|greatfakeman/Tabc...|79523|
+--------------------+-----+
```

# Which access keys are failing most often?

```
scala> val query="(SELECT access_key,COUNT(*) FROM ghtorrent WHERE access_key<>'' AND information LIKE '%Failed%' GROUP BY access_key ORDER BY COUNT(*) DESC LIMIT 1) as query"
query: String = (SELECT access_key,COUNT(*) FROM ghtorrent WHERE access_key<>'' AND information LIKE '%Failed%' GROUP BY access_key ORDER BY COUNT(*) DESC LIMIT 1) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [access_key: string, count: bigint]

scala> personDf.show()
+-----------+-----+
| access_key|count|
+-----------+-----+
|ac6168f8776|79623|
+-----------+-----+
```

# Setting 1: one executor

# Compute the number of different repositories accessed by the client ghtorrent-22 .Note the time taken by your query.

```
scala> val query="(SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query"
query: String = (SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]
```

```
ghtorrent=# CREATE INDEX idx on ghtorrent(downloader_id);
CREATE INDEX
```

```
scala> spark.time(personDf.show())
+-----+
|count|
+-----+
| 9041|
+-----+

Time taken: 6350 ms
```

# Now drop your index and compute the number of different repositories accessed by the client ghtorrent-22. Note the time taken by your query now.

```
scala> val query="(SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query"
query: String = (SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]
```

```
ghtorrent=# DROP INDEX idx;
DROP INDEX
```

```
scala> spark.time(personDf.show())
+-----+
|count|
+-----+
| 9041|
+-----+

Time taken: 7391 ms
```

# Setting 2: 2 executors

```
scala> spark.conf.set("spark.executor.instances",2)
```

# Compute the number of different repositories accessed by the client ghtorrent-22 .Note the time taken by your query.

```
scala> val query="(SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query"
query: String = (SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]
```

```
ghtorrent=# CREATE INDEX idx on ghtorrent(downloader_id);
CREATE INDEX
```

```
scala> spark.time(personDf.show())
+-----+
|count|
+-----+
| 9041|
+-----+

Time taken: 5478 ms
```

# Now drop your index and compute the number of different repositories accessed by the client ghtorrent-22. Note the time taken by your query now.

```
scala> val query="(SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query"
query: String = (SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22') as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]
```

```
ghtorrent=# DROP INDEX idx;
DROP INDEX
```

```
scala> spark.time(personDf.show())
+-----+
|count|
+-----+
| 9041|
+-----+

Time taken: 6268 ms
```

# Read in the CSV file into another table call it interesting. How many records are there?

```
scala> val query="(select count(*) from interesting) as query"
query: String = (select count(*) from interesting) as query

scala> val query="(select count(*) from interesting) as query"
query: String = (select count(*) from interesting) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]

scala> personDf.show()
+-----+
|count|
+-----+
| 1435|
+-----+
```

# How many records in the log file refer to entries in the interesting file?

```
scala> val query="(SELECT COUNT(*) FROM ghtorrent JOIN interesting ON repo=url) as query"
query: String = (SELECT COUNT(*) FROM ghtorrent JOIN interesting ON repo=url) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [count: bigint]

scala> personDf.show()
+-----+
|count|
+-----+
|  535|
+-----+
```

# Which of the interesting repositories has the most failed API calls?

```
scala> val query="(SELECT repo,COUNT(*) FROM ghtorrent JOIN interesting ON repo=url WHERE information LIKE '%Failed%' GROUP BY repo ORDER BY COUNT(*) DESC LIMIT 5) as query"
query: String = (SELECT repo,COUNT(*) FROM ghtorrent JOIN interesting ON repo=url WHERE information LIKE '%Failed%' GROUP BY repo ORDER BY COUNT(*) DESC LIMIT 5) as query

scala> val personDf = spark.read.jdbc(url, query, connectionProperties)
personDf: org.apache.spark.sql.DataFrame = [repo: string, count: bigint]

scala> personDf.show()
+--------------------+-----+
|                repo|count|
+--------------------+-----+
|  EndFirstCorp/onedb|    1|
|cendrine-b/la-lou...|    1|
|        zetaops/riak|    1|
+--------------------+-----+
```

# Part 2: MongoDB with Apache Spark

```
scala>  import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> val spark = SparkSession.builder().master("local").appName("MongoSparkConnectorIntro").config("spark.mongodb.input.uri", "mongodb://localhost:27017/BDA_2.ghtorrent").config("spark.mongodb.output.uri", "mongodb://localhost:27017/BDA_2.ghtorrent").getOrCreate()
20/02/12 22:21:01 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
20/02/12 22:21:01 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@359f4427
```

# How many records does the table contain?

```
scala> val query = spark.sql("SELECT COUNT(log_level) FROM ghtorrent")
query: org.apache.spark.sql.DataFrame = [count(log_level): bigint]

scala> query.show()
+----------------+
|count(log_level)|
+----------------+
|         9669634|
+----------------+
```

# Count the number of WARNing messages.

```
scala> val query = spark.sql("SELECT COUNT(*) FROM ghtorrent WHERE log_level='WARN'")
query: org.apache.spark.sql.DataFrame = [count(1): bigint]

scala> query.show()
+--------+
|count(1)|
+--------+
|  132158|
+--------+
```

# How many repositories were processed in total?

```
scala> val query = spark.sql("SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE repo <>'' AND information LIKE '%URL%repos%/%?%' AND ret_stage='api_client.rb'")
query: org.apache.spark.sql.DataFrame = [count(DISTINCT repo): bigint]

scala> query.show()
+--------------------+
|count(DISTINCT repo)|
+--------------------+
|              381194|
+--------------------+
```

# Which 10 clients did the highest HTTP requests?

```
scala> val query = spark.sql("SELECT downloader_id,COUNT(*) FROM ghtorrent WHERE information LIKE '%URL: https%' GROUP BY downloader_id ORDER BY COUNT(*) DESC LIMIT 10")
query: org.apache.spark.sql.DataFrame = [downloader_id: string, count(1): bigint]

scala> query.show()
+-------------+--------+
|downloader_id|count(1)|
+-------------+--------+
| ghtorrent-13|   85528|
|  ghtorrent-4|   19046|
| ghtorrent-18|   18948|
| ghtorrent-10|   18926|
| ghtorrent-40|   18911|
| ghtorrent-39|   18616|
| ghtorrent-38|   18614|
| ghtorrent-47|   18604|
|  ghtorrent-1|   18463|
| ghtorrent-24|   18452|
+-------------+--------+
```

# Which 10 client did the highest FAILED HTTP requests? (Operation part starts with the string ”Failed”)

```
scala> val query = spark.sql("SELECT downloader_id,COUNT(*) FROM ghtorrent WHERE information LIKE '%Failed%URL: https%' GROUP BY downloader_id ORDER BY COUNT(*) DESC LIMIT 10")
query: org.apache.spark.sql.DataFrame = [downloader_id: string, count(1): bigint]

scala> query.show()
+-------------+--------+
|downloader_id|count(1)|
+-------------+--------+
| ghtorrent-13|   79623|
| ghtorrent-21|    1378|
| ghtorrent-40|    1134|
| ghtorrent-18|     368|
| ghtorrent-42|     357|
|  ghtorrent-9|     356|
|  ghtorrent-4|     352|
| ghtorrent-25|     342|
| ghtorrent-22|     333|
|  ghtorrent-6|     332|
+-------------+--------+
```

# What is the most active hour of day?

```
scala> val query = spark.sql("SELECT hour,COUNT(*) FROM ghtorrent GROUP BY hour ORDER BY COUNT(*) DESC LIMIT 1")
query: org.apache.spark.sql.DataFrame = [hour: string, count(1): bigint]

scala> query.show()
+----+--------+
|hour|count(1)|
+----+--------+
|  10| 2662487|
+----+--------+
```

# What is the most active repository?

```
scala> val query = spark.sql("SELECT repo,COUNT(*) FROM ghtorrent WHERE repo<>'' GROUP BY repo ORDER BY COUNT(*) DESC LIMIT 1")
query: org.apache.spark.sql.DataFrame = [repo: string, count(1): bigint]

scala> query.show()
+--------------------+--------+
|                repo|count(1)|
+--------------------+--------+
|greatfakeman/Tabc...|   79523|
+--------------------+--------+
```

# Which access keys are failing most often?

```
scala> val query = spark.sql("SELECT access_key,COUNT(*) FROM ghtorrent WHERE access_key<>'' AND information LIKE '%Failed%' GROUP BY access_key ORDER BY COUNT(*) DESC LIMIT 1")
query: org.apache.spark.sql.DataFrame = [access_key: string, count(1): bigint]

scala> query.show()
+-----------+--------+
| access_key|count(1)|
+-----------+--------+
|ac6168f8776|   79623|
+-----------+--------+
```

# Setting 1: one executor

# Compute the number of different repositories accessed by the client ghtorrent-22 .Note the time taken by your query.

```
scala> val query = spark.sql("SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22'")
query: org.apache.spark.sql.DataFrame = [count(DISTINCT repo): bigint]
```

index_1

downloader_id | REGULAR | 51.4 MB | 0 since Wed Feb 12 2020

```
scala> spark.time(query.show())
+--------------------+
|count(DISTINCT repo)|
+--------------------+
|                9041|
+--------------------+

Time taken: 5925 ms
```

# Now drop your index and compute the number of different repositories accessed by the client ghtorrent-22. Note the time taken by your query now.

```
scala> val query = spark.sql("SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22'")
query: org.apache.spark.sql.DataFrame = [count(DISTINCT repo): bigint]

scala> spark.time(query.show())
+--------------------+
|count(DISTINCT repo)|
+--------------------+
|                9041|
+--------------------+

Time taken: 17633 ms
```

# Setting 2: 2 executors

```
scala> spark.conf.set("spark.executor.instances",2)
```

# Compute the number of different repositories accessed by the client ghtorrent-22 .Note the time taken by your query.

```
scala> val query = spark.sql("SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22'")
query: org.apache.spark.sql.DataFrame = [count(DISTINCT repo): bigint]
```

index_1

downloader_id

REGULAR

51.4 MB

0 since Wed Feb 12 2020

```
scala> spark.time(query.show())
+--------------------+
|count(DISTINCT repo)|
+--------------------+
|                9041|
+--------------------+

Time taken: 5187 ms
```

# Now drop your index and compute the number of different repositories accessed by the client ghtorrent-22. Note the time taken by your query now.

```
scala> val query = spark.sql("SELECT COUNT(DISTINCT(repo)) FROM ghtorrent WHERE downloader_id='ghtorrent-22'")
query: org.apache.spark.sql.DataFrame = [count(DISTINCT repo): bigint]
```

```
scala> spark.time(query.show())
+--------------------+
|count(DISTINCT repo)|
+--------------------+
|                9041|
+--------------------+

Time taken: 14931 ms
```

# Read in the CSV file into another table call it interesting. How many records are there?

```
scala> val df2 = MongoSpark.load(spark2)
df2: org.apache.spark.sql.DataFrame = [_id: struct<oid: string>, created_at: string ... 8 more fields]

scala> df2.createOrReplaceTempView("interesting")

scala> val query = spark2.sql("SELECT COUNT(*) FROM interesting")
query: org.apache.spark.sql.DataFrame = [count(1): bigint]

scala> query.show()
+--------+
|count(1)|
+--------+
|    1435|
+--------+
```

# How many records in the log file refer to entries in the interesting file?

```
scala> df.createOrReplaceTempView("ghtorrent")

scala> df3.createOrReplaceTempView("interesting")

scala> val query = spark.sql("SELECT COUNT(*) FROM ghtorrent JOIN interesting ON repo=url")
query: org.apache.spark.sql.DataFrame = [count(1): bigint]

scala> query.show()
+--------+
|count(1)|
+--------+
|     535|
+--------+
```

# Which of the interesting repositories has the most failed API calls?

```
scala> val query = spark.sql("SELECT repo,COUNT(*) FROM ghtorrent JOIN interesting ON repo=url WHERE information LIKE '%Failed%' GROUP BY repo ORDER BY COUNT(*) DESC LIMIT 5")
query: org.apache.spark.sql.DataFrame = [repo: string, count(1): bigint]

scala> query.show()
+--------------------+--------+
|                repo|count(1)|
+--------------------+--------+
|cendrine-b/la-lou...|       1|
|        zetaops/riak|       1|
|  EndFirstCorp/onedb|       1|
+--------------------+--------+
```

# Part 3: HDFS with Apache Spark

```
Priyas-MacBook-Pro:sbin priyarajpurohit$ hdfs dfs -put /Users/priyarajpurohit/Desktop/ghtorrent-logs.csv  hdfs://localhost:9000/user/hduser
2020-02-11 22:05:31,788 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2020-02-11 22:05:32,536 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:33,102 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:33,639 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:34,122 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:34,604 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:35,074 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:35,541 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:36,015 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:36,457 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2020-02-11 22:05:36,989 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
Priyas-MacBook-Pro:sbin priyarajpurohit$
```

```
scala> import java.text.SimpleDateFormat
import java.text.SimpleDateFormat

scala> import java.util.Date
import java.util.Date

scala> case class LogLine(debug_level: String, timestamp: Date, download_id: Integer,
     |                    retrieval_stage: String, rest: String);
defined class LogLine

scala>

scala> val dateFormat = "yyyy-MM-dd:HH:mm:ss"
dateFormat: String = yyyy-MM-dd:HH:mm:ss

scala> val regex = """([^\s]+), ([^\s]+)\+00:00, ghtorrent-([^\s]+) -- ([^\s]+).rb: (.*$)""".r
regex: scala.util.matching.Regex = ([^\s]+), ([^\s]+)\+00:00, ghtorrent-([^\s]+) -- ([^\s]+).rb: (.*$)

scala> val rdd = sc.
     |     textFile("hdfs://localhost:9000/user/hduserghtorrent-logs.txt")
rdd: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/user/hduserghtorrent-logs.txt MapPartitionsRDD[1] at textFile at <console>:27

scala> val rdd = sc.
     |     textFile("hdfs://localhost:9000/user/hduser/ghtorrent-logs.txt").
     | flatMap ( x => x match {
     |       case regex(debug_level, dateTime, downloadId, retrievalStage, rest) =>
     |         val df = new SimpleDateFormat(dateFormat)
     |         new Some(LogLine(debug_level, df.parse(dateTime.replace("T", ":")), downloadId.toInt, retrievalStage, rest))
     |       case _ => None;
     |       })
rdd: org.apache.spark.rdd.RDD[LogLine] = MapPartitionsRDD[4] at flatMap at <console>:34
```

# How many records does the table contain?

```
scala> rdd.count
res0: Long = 9669634
```

# Count the number of WARNing messages.

```
scala> rdd.filter(x => x.debug_level == "WARN").count
res2: Long = 132158
```

# How many repositories were processed in total?

```
scala> val repos = rdd.filter(_.retrieval_stage == "api_client").map(_.rest.split("/").slice(4,6).mkString("/").takeWhile(_ != '?'))
repos: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[8] at map at <console>:27

scala> repos.distinct.count
res3: Long = 78589
```

# Which 10 clients did the highest HTTP requests?

```
scala> rdd.filter(_.retrieval_stage == "api_client").
     |     keyBy(_.download_id).
     |     mapValues(1 => 1).
     |     reduceByKey((a,b) => a + b).
     |     sortBy(x => x._2, false).
     |     take(10)
res4: Array[(Integer, Int)] = Array((13,135978), (21,100906), (20,31401), (40,30774), (42,27631), (35,26131), (2,26075), (47,25105), (46,24691), (4,24639))
```

# Which 10 client did the highest FAILED HTTP requests? (Operation part starts with the string ”Failed”)

```
scala> rdd.filter(_.retrieval_stage == "api_client").
     |     filter(_.rest.startsWith("Failed")).
     |     keyBy(_.download_id).
     |     mapValues(l => 1).
     |     reduceByKey((a,b) => a + b).
     |     sortBy(x => x._2, false).
     |     take(10)
res5: Array[(Integer, Int)] = Array((13,79623), (21,1378), (40,1134), (18,368), (42,357), (9,356), (4,352), (25,342), (22,333), (6,332))
```

# What is the most active hour of day?

```
scala> rdd.keyBy(_.timestamp.getHours).
     |     mapValues(l => 1).
     |     reduceByKey((a,b) => a + b).
     |     sortBy(x => x._2, false).
     |     take(1)
warning: there was one deprecation warning; re-run with -deprecation for details
res6: Array[(Int, Int)] = Array((10,2662487))

scala>
```

# What is the most active repository?

```
scala> repos.
     |     filter(_.nonEmpty).
     |     map(x => (x, 1)).
     |     reduceByKey((a,b) => a + b).
     |     sortBy(x => x._2, false).
     |     take(1)
res7: Array[(String, Int)] = Array((greatfakeman/Tabchi,79524))

scala>
```

# Setting 1: one executor

# Which access keys are failing most often?

```
scala> val start_time = System.currentTimeMillis();
start_time: Long = 1581442450075

scala> rdd.filter(_.rest.startsWith("Failed")).
     |   filter(_.rest.contains("Access: ")).
     |   map(_.rest.split("Access: ", 2)(1).split(",", 2)(0)).
     |   map(x => (x, 1)).
     |   reduceByKey((a,b) => a + b).
     |   sortBy(x => x._2, false).
     |   take(1)
res0: Array[(String, Int)] = Array((ac6168f8776,79623))

scala>

scala> println("Took " + (System.currentTimeMillis() - start_time) + "ms.")
Took 19923ms.
```

# Setting 2: 2 executors

```
scala> spark.conf.set("spark.executor.instances",2)
```

# Which access keys are failing most often?

```
scala> val start_time = System.currentTimeMillis();
start_time: Long = 1581442549615

scala> rdd.filter(_.rest.startsWith("Failed")).
     |   filter(_.rest.contains("Access: ")).
     |   map(_.rest.split("Access: ", 2)(1).split(",", 2)(0)).
     |   map(x => (x, 1)).
     |   reduceByKey((a,b) => a + b).
     |   sortBy(x => x._2, false).
     |   take(1)
res3: Array[(String, Int)] = Array((ac6168f8776,79623))

scala>

scala> println("Took " + (System.currentTimeMillis() - start_time) + "ms.")
Took 18330ms.
```

# Read in the CSV file into another table call it interesting. How many records are there?

```
scala> val interesting = sc.
     |     textFile("hdfs://localhost:9000/user/hduser/important-repos.csv").
     |     mapPartitionsWithIndex((idx, iter) => if (idx == 0) iter.drop(1) else iter). //remove header from CSV file
     |     flatMap ( x => x match {
     |       case regex(id, url, owner_id, name, language, created_at, forked_from, deleted, updated_at) => {
     |         val df = new SimpleDateFormat(dateFormat)
     |         new Some(Repo(id.toInt, url, owner_id.toInt, name, language, df.parse(created_at), forked_from, deleted.toInt, df.parse(updated_at)))
     |       }
     |       case _ => print(x); None;
     |       }).cache()
interesting: org.apache.spark.rdd.RDD[Repo] = MapPartitionsRDD[34] at flatMap at <console>:35

scala> interesting.count
res7: Long = 1435
```

# How many records in the log file refer to entries in the interesting file?

```
res7: Long = 1455

scala> val interestingRepo = interesting.keyBy(_.name);
interestingRepo: org.apache.spark.rdd.RDD[(String, Repo)] = MapPartitionsRDD[35] at keyBy at <console>:27

scala> val logLineRepo = rdd.keyBy(_.rest).
     |                       map(x => x.copy(_1 = x._1.split("/").slice(4,6).mkString("/").takeWhile(_ != '?').split("/", 2).last)).
     |                       filter(_._1.nonEmpty); //delete all empty repos
logLineRepo: org.apache.spark.rdd.RDD[(String, LogLine)] = MapPartitionsRDD[38] at filter at <console>:29

scala>

scala> val joinedRepo = interestingRepo.join(logLineRepo);
joinedRepo: org.apache.spark.rdd.RDD[(String, (Repo, LogLine))] = MapPartitionsRDD[41] at join at <console>:29

scala> joinedRepo.count;
res8: Long = 87930
```

# Which of the interesting repositories has the most failed API calls?

```
scala> joinedRepo.filter(x => x._2._2.rest.startsWith("Failed")).
     |            map(x => (x._1, 1)).
     |            reduceByKey((a,b) => a + b).
     |            sortBy(x => x._2, false).
     |            take(3)
res9: Array[(String, Int)] = Array((hello-world,740), (test,309), (demo,166))
```

# Challenges

- Can’t upload a csv file to MongoDB directly
  - Convert the csv file into json
- Multiple SparkSessions can’t run in the same shell/two shells at the same time
  - Use spark.stop

# Learning

- The methods of connecting Apache Spark to each application used is different
- Learnt a new language-Scala
- Got familiar with the usage and storage of data for the 3 applications

# Resources Used

PostgreSQL Documentation
MongoDB, MongoDB-spark Connector Documentation
Hadoop HDFS Documentation
Spark Documentation
Postgres.app documentation
Lecture Slides(SQL)