

Final-Submission – Logic Explanation

Explanation of the solution to the streaming layer problem

***Please zoom to 180% or 200% to see screenshots with better clarity

1. In order to complete below tasks, I have created EMR cluster with **HBase** , **Hadoop** , **Hive** , **Hue** , **jupyterhub** , **Livy** , **spark** , **Sqoop** and **Zookeeper** , Root device EBS volume size as 20 GB

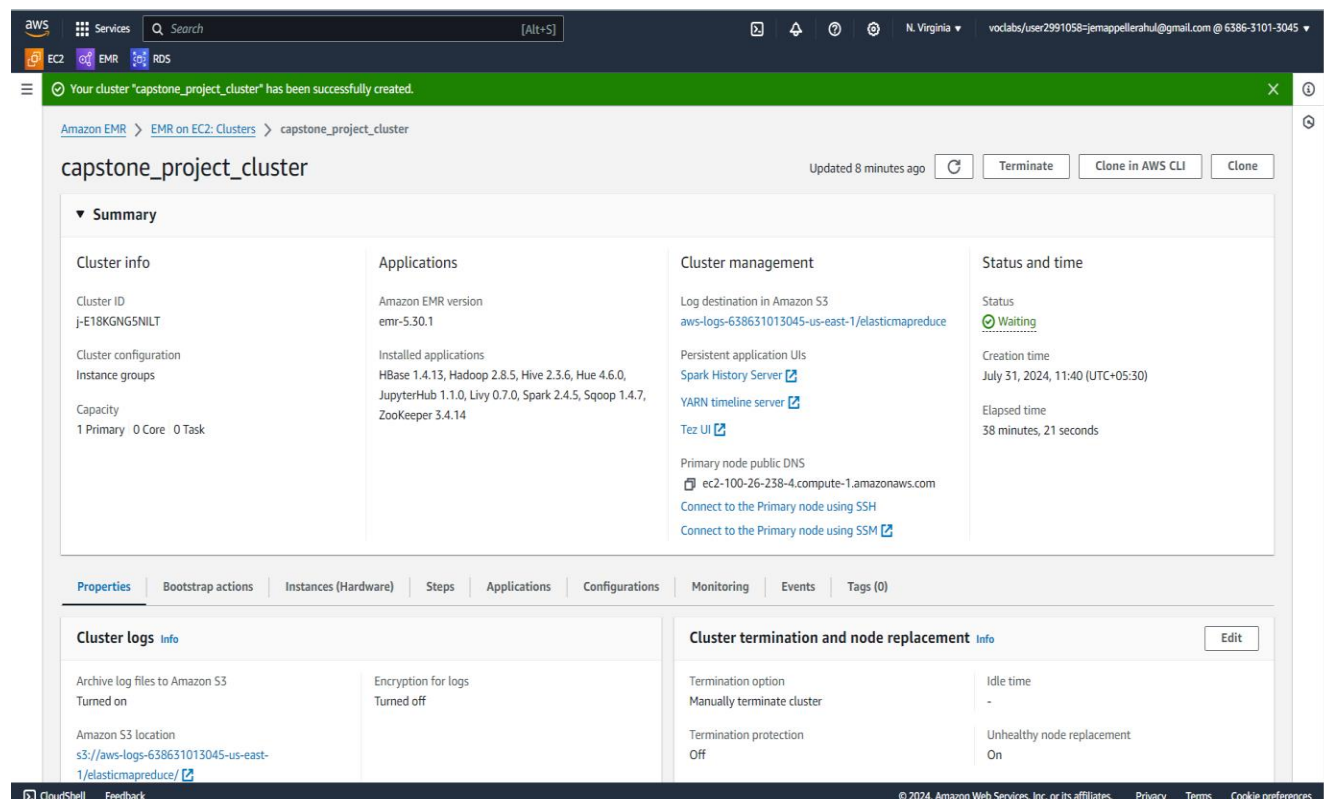
• **Task 5:** Create a streaming data processing framework that ingests real-time POS transaction data from

Kafka. The transaction data is then validated based on the three rules' parameters (stored in the NoSQL database) discussed previously.

• **Task 6:** Update the transactions data along with the status (fraud/genuine) in the card_transactions table.

• **Task 7:** Store the 'postcode' and 'transaction_dt' of the current transaction in the look-up table in the NoSQL database if the transaction was classified as genuine.

EMR Cluster Configuration:



The screenshot displays the AWS Management Console interface for an Amazon EMR cluster named 'capstone_project_cluster'. The cluster is in the 'Waiting' state and was updated 8 minutes ago. The console shows various configuration details and tabs for managing the cluster.

Summary			
Cluster info	Applications	Cluster management	Status and time
Cluster ID j-E18KNG5NILT Cluster configuration Instance groups Capacity 1 Primary 0 Core 0 Task	Amazon EMR version emr-5.30.1 Installed applications HBase 1.4.13, Hadoop 2.8.5, Hive 2.3.6, Hue 4.6.0, JupyterHub 1.1.0, Livy 0.7.0, Spark 2.4.5, Sqoop 1.4.7, ZooKeeper 3.4.14	Log destination in Amazon S3 aws-logs-638631013045-us-east-1/elasticmapreduce Persistent application Uls Spark History Server YARN timeline server Tez UI Primary node public DNS ec2-100-26-238-4.compute-1.amazonaws.com Connect to the Primary node using SSH Connect to the Primary node using SSM	Status Waiting Creation time July 31, 2024, 11:40 (UTC+05:30) Elapsed time 38 minutes, 21 seconds

Cluster logs		Cluster termination and node replacement	
Archive log files to Amazon S3 Turned on Amazon S3 location s3://aws-logs-638631013045-us-east-1/elasticmapreduce/	Encryption for logs Turned off	Termination option Manually terminate cluster Termination protection Off	Idle time - Unhealthy node replacement On

2. Logged into EMR instance as “hadoop”:

```
login as: hadoop
Authenticating with public key "rahulskey"

      _|_  _|_  )
     _|_ ( _|_ /  Amazon Linux 2 AMI
    _|_ \ _|_ _|_

https://aws.amazon.com/amazon-linux-2/

EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRRRRRRRRRRR
E::::::::::::::::::::E M::::::::M M::::::::M R:::::::::R
EE::::::::EEEEEEEE::::E M::::::::M M::::::::M R::::RRRRRR::::R
  E:::E      EEEEE M::::::::M M::::::::M RR:::R      R:::R
  E:::E      M::::M M:::M M:::M M:::M R:::R      R:::R
  E:::EEEEEEEEEE M:::M M:::M M:::M M:::M R:::RRRRRR::::R
  E::::::::::::E M:::M M:::M M:::M M:::M R:::::::::RR
  E:::EEEEEEEEEE M:::M M:::M M:::M M:::M R:::RRRRRR::::R
  E:::E      M:::M M:::M M:::M M:::M R:::R      R:::R
  E:::E      EEEEE M:::M M:::M M:::M M:::M R:::R      R:::R
EE::::::::EEEEEEEE::::E M:::M M:::M M:::M R:::R      R:::R
E::::::::::::E M:::M M:::M M:::M M:::M RR:::R      R:::R
EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRRR RRRRRR
```

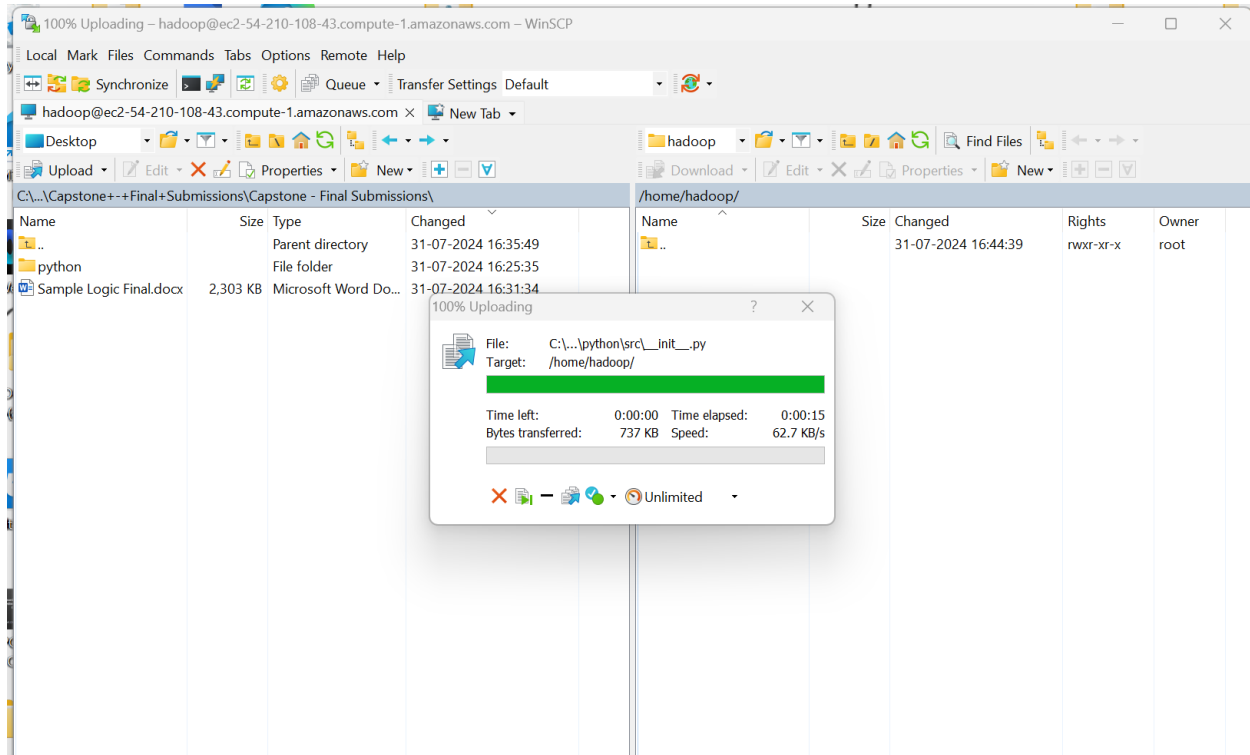
3. Switch to root user and run **pip install kafka-python** and then again use “**sudo -i -u hadoop**” to be a hadoop user

```
[root@ip-172-31-49-181 hadoop]# pip install kafka-python
WARNING: Running pip install with root privileges is generally not a good idea. Try 'pip3 install --user' instead.
Collecting kafka-python
  Downloading https://files.pythonhosted.org/packages/75/68/dcb0db055309f680ab2931a3eeb22d065604b638acf0c914bedf4c1a0c0c/kafka_python-2.0.2-py2.py3-none-any.whl (246kB)
    100% |#####| 256kB 3.8MB/s
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
[root@ip-172-31-49-181 hadoop]#
```

4. Run the following commands in order to Install Happy base and start thrift server

- **sudo yum update**
- **sudo yum install python3-devel**
- **pip install happybase**
- **/usr/lib/hbase/bin/hbase-daemon.sh start thrift -p 9090**

5. Downloaded db-> **dao.py** , **geomap.py** ,**rules-> rules.py** ,**driver.py** ,**unzipsv.csv** from the resource section of the capstone project from the learning platform and transfer it to hadoop instance via WinSCP.



6. Updated the Public IP of your EC2 Instance “100.26.238.4”(self.host) in **dao.py** file

```
1  import happybase
2
3  class HBaseDao:
4      """
5      Dao class for operation on HBase
6      """
7      __instance = None
8
9      @staticmethod
10     def get_instance():
11         """ Static access method. """
12         if HBaseDao.__instance == None:
13             HBaseDao()
14         return HBaseDao.__instance
15
16     def __init__(self):
17         if HBaseDao.__instance != None:
18             raise Exception("This class is a singleton!")
19         else:
20             HBaseDao.__instance = self
21             self.host = '100.26.238.4'
22             #self.host = 'localhost'
23             for i in range(2):
```

7. Updated rules.py with following parameters:

lookup_table = 'lookup_data_hbase'

master_table = 'card_transactions_hbase'

```
# List all the functions to check for the rules
from db.dao import HBaseDao
from db.geo_map import GEO_Map
from datetime import datetime
import uuid

# Create UDF functions
lookup_table = 'lookup_data_hbase'
master_table = 'card_transactions_hbase'
```

8. Created Python functions, containing the logic for the UDFs (rules.py)

verify_ucl_data : Function to verify the UCL rule Transaction amount should be less than Upper control limit (UCL)

```
def verify_ucl_data(card_id, amount):
    try:
        hbasedao = HBaseDao.get_instance()

        card_row = hbasedao.get_data(key=str(card_id), table=lookup_table)
        card_ucl = (card_row[b'card_data:ucl']).decode("utf-8")

        if amount < float(card_ucl):
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

verify_credit_score_data: Function to verify the credit score rule .Credit score of each member should be greater than 200

```
def verify_credit_score_data(card_id):

    try:
        hbasedao = HBaseDao.get_instance()

        card_row = hbasedao.get_data(key=str(card_id), table=lookup_table)
        card_score = (card_row[b'card_data:score']).decode("utf-8")

        if int(card_score) > 200:
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

verify_postcode_data: Function to verify the following zipcode rules. ZIP code distance

```
def verify_postcode_data(card_id, postcode, transaction_dt):

    try:
        hbasedao = HBaseDao.get_instance()
        geo_map = GEO_Map.get_instance()

        card_row = hbasedao.get_data(key=str(card_id), table=lookup_table)
        last_postcode = (card_row[b'card_data:postcode']).decode("utf-8")
        last_transaction_dt = (card_row[b'card_data:transaction_dt']).decode("utf-8")

        current_lat = geo_map.get_lat(str(postcode))
        current_lon = geo_map.get_long(str(postcode))
        previous_lat = geo_map.get_lat(last_postcode)
        previous_lon = geo_map.get_long(last_postcode)

        dist = geo_map.distance(lat1=current_lat, long1=current_lon, lat2=previous_lat, long2=previous_lon)

        speed = calculate_speed(dist, transaction_dt, last_transaction_dt)

        if speed < speed_threshold:
            return True
        else:
            return False
    except Exception as e:
        raise Exception(e)
```

calculate_speed : A function to calculate the speed from distance and transaction timestamp differentials

```
def calculate_speed(dist, transaction_dt1, transaction_dt2):

    transaction_dt1 = datetime.strptime(transaction_dt1, '%d-%m-%Y %H:%M:%S')
    transaction_dt2 = datetime.strptime(transaction_dt2, '%d-%m-%Y %H:%M:%S')

    elapsed_time = transaction_dt1 - transaction_dt2
    elapsed_time = elapsed_time.total_seconds()

    try:
        return dist / elapsed_time
    except ZeroDivisionError:
        return 299792.458
# (Speed of light)
```

verify_rules_status: A function to verify all the three rules - ucl, credit score and speed

```
def verify_rules_status(card_id, member_id, amount, pos_id, postcode, transaction_dt):

    hbasedao = HBaseDao.get_instance()

    # Check if the POS transaction passes all rules.
    # If yes, update the lookup table and insert data in master table as genuine.
    # Else insert the transaction in master table as Fraud.

    rule1 = verify_ucl_data(card_id, amount)
    rule2 = verify_credit_score_data(card_id)
    rule3 = verify_postcode_data(card_id, postcode, transaction_dt)

    if all([rule1, rule2, rule3]):
        status = 'GENUINE'
        hbasedao.write_data(key=str(card_id),
                            row={'card_data:postcode': str(postcode), 'card_data:transaction_dt': str(transaction_dt)},
                            table=lookup_table)
    else:
        status = 'FRAUD'

    new_id = str(uuid.uuid4()).replace('-', '')
    hbasedao.write_data(key=new_id,
                        row={'cardDetail:card_id': str(card_id), 'cardDetail:member_id': str(member_id),
                            'transactionDetail:amount': str(amount), 'transactionDetail:pos_id': str(pos_id),
                            'transactionDetail:postcode': str(postcode), 'transactionDetail:status': str(status),
                            'transactionDetail:transaction_dt': str(transaction_dt)},
                        table=master_table)

    return status
```

9. Next, I updated the **'driver.py'** file with the following code Setting up the system dependencies and importing necessary libraries and modules

```
import os
import sys
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from rules.rules import *
```

10. Initializing the Spark session and reading input data from Kafka mentioning the details of the Kafka broker, such as bootstrap server, port and topic name

1. Connect to kafka topic using

Bootstrap-server: 18.211.252.152

Port Number: 9092

Topic: transactions-topic-verified

```
# Initialize Spark session
spark = SparkSession.builder.appName("CreditCardFraud").getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Read stream from Kafka
credit_data = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("startingOffsets", "earliest") \
    .option("failOnDataLoss", "false") \
    .option("subscribe", "transactions-topic-verified") \
    .load()
```

11. Define JSON schema of each transactions


```
# Define schema for transaction
dataSchema = StructType([
    StructField("card_id", LongType(), True),
    StructField("member_id", LongType(), True),
    StructField("amount", DoubleType(), True),
    StructField("pos_id", LongType(), True),
    StructField("postcode", IntegerType(), True),
    StructField("transaction_dt", StringType(), True)
])
```

12. Read the raw JSON data from Kafka as 'credit_data_stream' and Define UDF's to verify rules

```
# Extract parsed fields and filter nulls
credit_data_stream = parsed.select("credit_data.*") \
    .filter(
        col("card_id").isNotNull() &
        col("member_id").isNotNull() &
        col("amount").isNotNull() &
        col("postcode").isNotNull() &
        col("pos_id").isNotNull() &
        col("transaction_dt").isNotNull()
    )

# Define UDF which verifies all the rules for each transaction and updates the lookup and master tables
verify_all_rules = udf(verify_rules_status, StringType())

Final_data = credit_data_stream \
    .withColumn('status', verify_all_rules(credit_data_stream['card_id'],
        credit_data_stream['member_id'],
        credit_data_stream['amount'],
        credit_data_stream['pos_id'],
        credit_data_stream['postcode'],
        credit_data_stream['transaction_dt']))
```

13. Code to display output in console

```
# Write output to console as well
output_data = Final_data \
    .select("card_id", "member_id", "amount", "pos_id", "postcode", "transaction_dt") \
    .writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", False) \
    .start()
```

14. Define spark termination

```
# Indicating Spark to await termination
output_data.awaitTermination()
```

15. Set the Kafka Version using the following command

export SPARK_KAFKA_VERSION=0.10

16. Run the spark-submit command, specifying the Spark-SQL-Kafka package and python file and Check Output in console :

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 driver.py

```
hadoop@ip-172-31-50-180:~/python/src
xecutor.instances
24/07/31 06:44:14 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
24/07/31 06:44:14 INFO YarnSchedulerBackend$YarnSchedulerEndpoint: ApplicationMaster registered as NettyRpcEndpointRef(spark-client://YarnAM)
24/07/31 06:44:14 INFO YarnClientSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
24/07/31 06:44:14 INFO SharedState: loading hive config file: file:/etc/spark/conf/dist/hive-site.xml
24/07/31 06:44:14 INFO SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir ('hdfs://user/spark/warehouse').
24/07/31 06:44:14 INFO SharedState: Warehouse path is 'hdfs://user/spark/warehouse'.
24/07/31 06:44:15 INFO JettyUtils: Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /SQL.
24/07/31 06:44:15 INFO JettyUtils: Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /SQL/json.
24/07/31 06:44:15 INFO JettyUtils: Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /SQL/execution.
24/07/31 06:44:15 INFO JettyUtils: Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /SQL/execution/json.
24/07/31 06:44:15 INFO JettyUtils: Adding filter org.apache.hadoop.yarn.server.webproxy.amfilter.AmIpFilter to /static/sql.
24/07/31 06:44:16 INFO StateStoreCoordinatorRef: Registered StateStoreCoordinator endpoint

Batch: 0
-----
+-----+-----+-----+-----+-----+-----+-----+
|card_id|member_id|amount|pos_id|postcode|transaction_dt|status|
+-----+-----+-----+-----+-----+-----+-----+
|6492177672429642|856504383389409|3210949|676557552041346|30573|01-09-2018 03:14:17|GENUINE|
|6492177672429642|856504383389409|3426215|455088723035904|17583|04-06-2018 21:17:31|GENUINE|
|6492177672429642|856504383389409|7360323|80118552510202|32004|08-03-2018 09:21:55|FRAUD|
|6492177672429642|856504383389409|2550221|928148296374897|80648|28-05-2018 11:21:49|GENUINE|
|5186615811954262|856722666618984|7847676|164740339445208|62966|04-08-2018 00:09:24|FRAUD|
|5186615811954262|856722666618984|7814000|670906035098480|39335|12-11-2018 04:25:59|FRAUD|
|5186615811954262|856722666618984|8526457|949958430030766|18657|14-07-2018 00:16:31|FRAUD|
|5186615811954262|856722666618984|7400764|429648528231037|21719|16-11-2018 14:35:30|FRAUD|
|5186615811954262|856722666618984|3855127|499690126066748|87328|24-10-2018 13:22:08|GENUINE|
|5186615811954262|856722666618984|168989|128994049351990|16061|25-02-2018 19:10:54|GENUINE|
|5186615811954262|856722666618984|1487183|670381193676932|38643|29-07-2018 03:36:29|GENUINE|
|5186615811954262|856722666618984|2632731|901770477590614|71367|30-03-2018 09:55:02|GENUINE|
|5136319769899261|858469490248492|1292492|511340027319289|57382|12-02-2018 05:10:23|GENUINE|
|5136319769899261|858469490248492|5912087|97184584409061|23128|11-08-2018 08:35:24|FRAUD|
|5136319769899261|858469490248492|4513787|278416852864195|87018|11-10-2018 15:51:48|GENUINE|
|5136319769899261|858469490248492|9167631|639040185085953|21647|19-11-2018 06:53:07|FRAUD|
|5218975136635638|858500181320165|7947933|921773321870388|68055|06-04-2018 12:22:10|FRAUD|
|5218975136635638|858500181320165|7316931|260765159192906|18036|07-03-2018 17:39:56|FRAUD|
|5218975136635638|858500181320165|8850711|107928751019321|55073|07-08-2018 05:54:25|FRAUD|
|5218975136635638|858500181320165|3164357|822643805583880|14033|13-10-2018 20:11:43|GENUINE|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

Batch: 1
-----
+-----+-----+-----+-----+-----+-----+-----+
|card_id|member_id|amount|pos_id|postcode|transaction_dt|status|
+-----+-----+-----+-----+-----+-----+-----+
|4353614029446427|865787553237116|7348418|833286556889157|15047|13-10-2018 23:11:08|FRAUD|
+-----+-----+-----+-----+-----+-----+-----+
```

17 . Count Data in Hbase: count 'lookup_data_hive'

```
Current count: 34000, row: 5342400571435088~569800919443173~2016-08-26 07:22:56~2049055.0
Current count: 35000, row: 5380978184175608~667233655872426~2016-09-01 10:51:09~9057468.0
Current count: 36000, row: 5414439899219272~345244546468970~2017-06-07 00:10:15~1413268.0
Current count: 37000, row: 5481808794715436~234167732114687~2017-10-20 16:48:20~3012181.0
Current count: 38000, row: 5534323829711423~637125777249291~2017-06-30 11:22:05~535263.0
Current count: 39000, row: 5584977018799504~770952296643375~2017-05-03 09:46:23~9635501.0
Current count: 40000, row: 6011139413319542~936287997923127~2016-03-30 07:13:43~1908818.0
Current count: 41000, row: 6011525010455848~959531034098755~2017-08-02 23:46:55~2762867.0
Current count: 42000, row: 6011782857327719~167569302823461~2016-10-09 02:59:08~8187310.0
Current count: 43000, row: 6221796595498984~174952353920145~2016-02-18 04:11:46~7689490.0
Current count: 44000, row: 6224271253849917~048019406374284~2017-12-12 19:22:13~4960131.0
Current count: 45000, row: 6225606551069826~648178895653883~2017-07-13 13:58:12~4112360.0
Current count: 46000, row: 6228733641419063~422091580968713~2016-05-11 10:27:11~2641410.0
Current count: 47000, row: 6447877814927926~992747968210744~2018-01-11 00:00:00~9774152.0
Current count: 48000, row: 6461356425954109~839304530643246~2016-11-03 20:57:23~1877568.0
Current count: 49000, row: 6480152634975473~963893207999520~2018-01-17 17:22:08~5863997.0
Current count: 50000, row: 6505080237250161~874636482279942~2017-09-18 17:36:33~2156682.0
Current count: 51000, row: 6544876671165176~424648847023528~2017-08-04 19:11:48~5155210.0
Current count: 52000, row: 6574255180086418~504711588075355~2017-12-20 08:11:48~5899422.0
Current count: 53000, row: 6595814135833988~311764663134170~2017-11-18 12:30:42~2717739.0
Current count: 54000, row: 6461356425954109~839304530643246~03-11-2016 20:57:23~1877568.0
Current count: 55000, row: 6480152634975473~963893207999520~17-01-2018 17:22:08~5863997.0
Current count: 56000, row: 6505080237250161~874636482279942~18-09-2017 17:36:33~2156682.0
Current count: 57000, row: 651876671165176~12648817023528~01-08-2017 16:09:39~5155210.0
Current count: 58000, row: 6574255180086418~504711588075355~20-12-2017 08:11:48~5899422.0
Current count: 59000, row: 6595814135833988~311764663134170~18-11-2017 12:30:42~2717739.0
59367 row(s) in 2.1450 seconds

=> 59367

hbase (main) : 002:0>
```