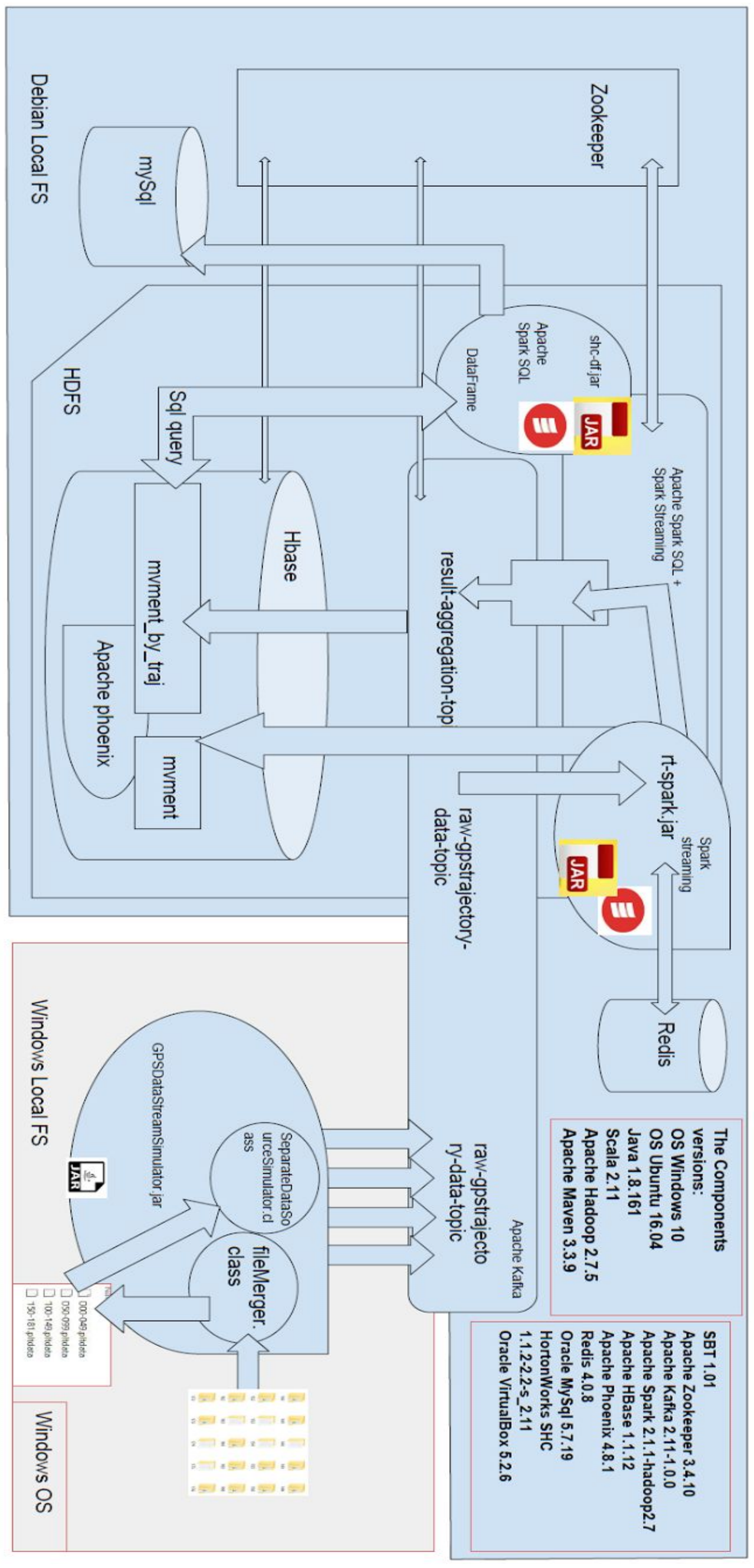
1. Project outline and used technologies

The project dedicates to the processing stream and non stream data on the technologies that enable by using the same architecture processing and analyzing Big Data. For the detailed description of dataset and project problem statement please refer to the pdf file ProblemStatement_Process_and_collect_GPS_trajectory_dataset.pdf
This document is dedicated to guiding through the code base of the project and point on the places on the code base where was implemented the solution to the problem which was imposed in ProblemStatement_Process_and_collect_GPS_trajectory_dataset.pdf for this purpose I integrate the marks like <<impl of solution to 3.0.1 ps>> which basically mean that in the nearest print screen you can find precise place where the solution of the subproblem was implemented.
At the end of this document, you can find the hyperlink to the video demos of data flow and jar output that prove in some degree that indeed the codebase can produce the needed result without bugs or exceptions. On the next page, you will find the architecture of the system with specification outline that I use the project solution flow. All solution was made on the customer made ubuntu OS (VM) with all installations steps which I do not describe in the document. If you will need some elaboration u can always contact me for further comments or consultations.

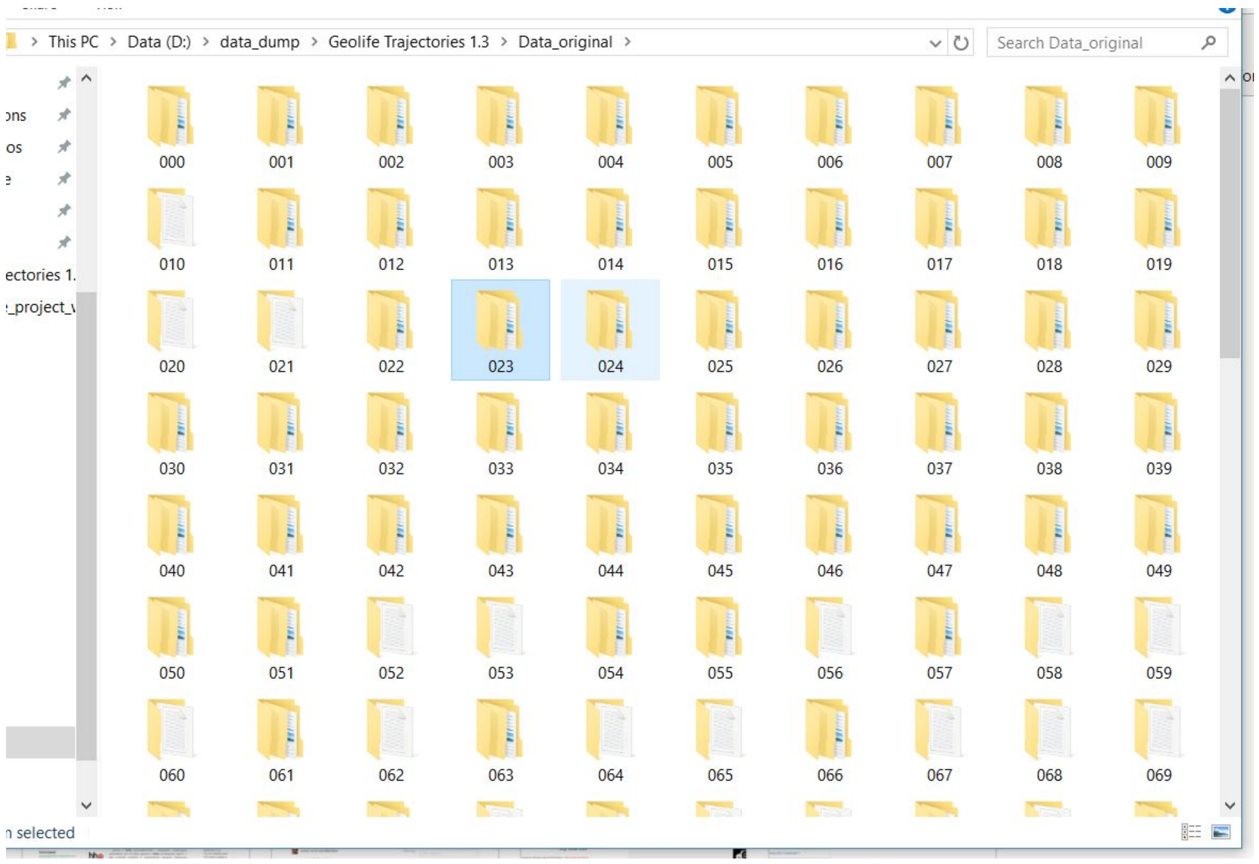
The components and technologies which was used :
OS Windows 10 - as a host OS for GPSDataStreamSimulator
OS Ubuntu 16.04 - used as a host OS for hadoop cluster
Apache Hadoop 2.7.5 - used as a host cluster for HBase
Apache Maven 3.3.9 - used as building tool for GPSDataStreamSimulator
SBT 1.01 - build tool for the shc-df.jar and rt-spark.jar
Apache Zookeeper 3.4.10 - used as a main synchronization between HBase , messaging broker Kafka, hadoop in general and etc
Apache Kafka 2.11-1.0.0 - used as a broker messaging system
Apache Spark 2.1.1-hadoop2.7- used for data analysis and writing to the HBase
Apache HBase 1.1.12 - NoSQL database for fast store and retrieval of data
Apache Phoenix 4.8.1 - for the monitoring and fast data analysis
Redis 4.0.8 - as a buffer component for calculating “dist” column and “tdiff”
Oracle MySql 5.7.19 - for post analysis storate of the analysis result
HortonWorks SHC 1.1.2-2.2-s_2.11 - for connection between apache spark and HBase to enable retrieving data as a DataFrame and enable running apache spark data analysis on the top of the YARN

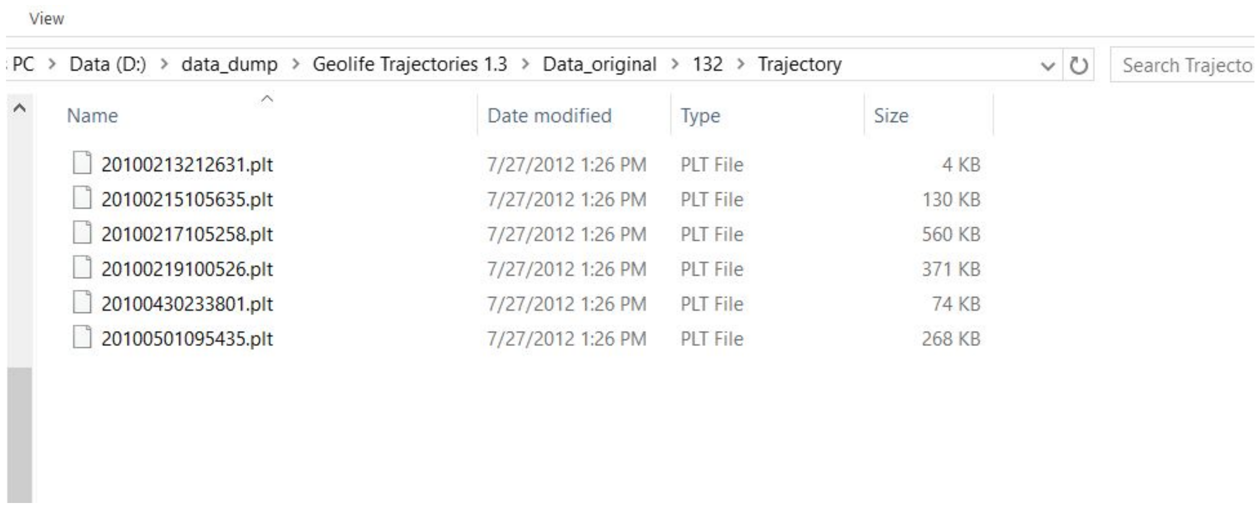
2. System Architecture description



3. Data Ingestion and initial Validation

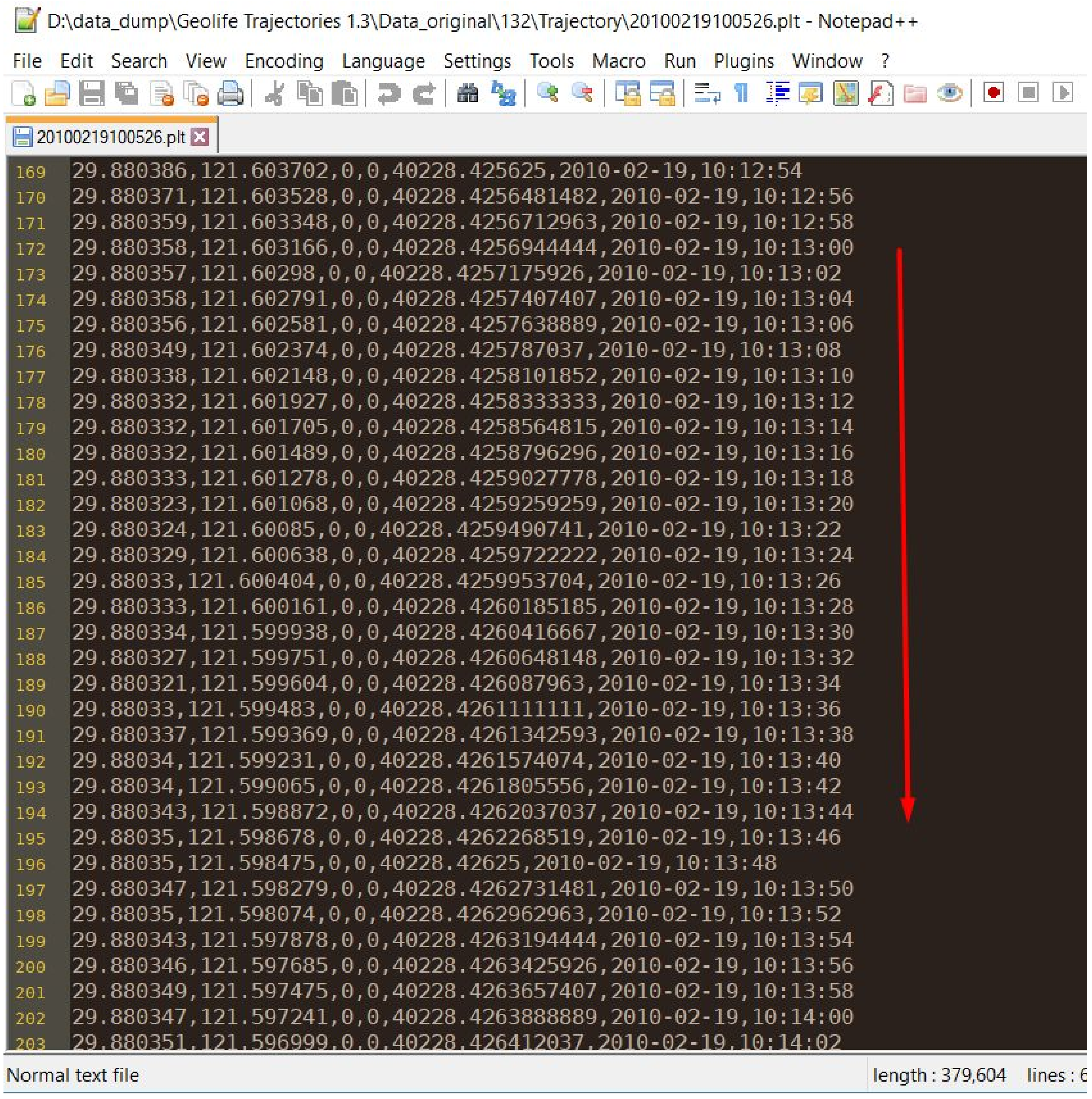
The original data contain 181 folders each correspond to the user in which several trajectories presented




View
PC > Data (D:) > data_dump > Geolife Trajectories 1.3 > Data_original > 132 > Trajectory
Search Trajecto

Name	Date modified	Type	Size
20100213212631.plt	7/27/2012 1:26 PM	PLT File	4 KB
20100215105635.plt	7/27/2012 1:26 PM	PLT File	130 KB
20100217105258.plt	7/27/2012 1:26 PM	PLT File	560 KB
20100219100526.plt	7/27/2012 1:26 PM	PLT File	371 KB
20100430233801.plt	7/27/2012 1:26 PM	PLT File	74 KB
20100501095435.plt	7/27/2012 1:26 PM	PLT File	268 KB

So the first challenge that we should overcome is to somehow merge all these files into 1 - 8 corresponding how many cores we wanna dedicate to streaming data. Also in the big files the chronological order should be maintained as in original one in order to make possible to deliver data enrichment in our system.


D:\data_dump\Geolife Trajectories 1.3\Data_original\132\Trajectory\20100219100526.plt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
20100219100526.plt
Normal text file
length : 379,604 lines : 6

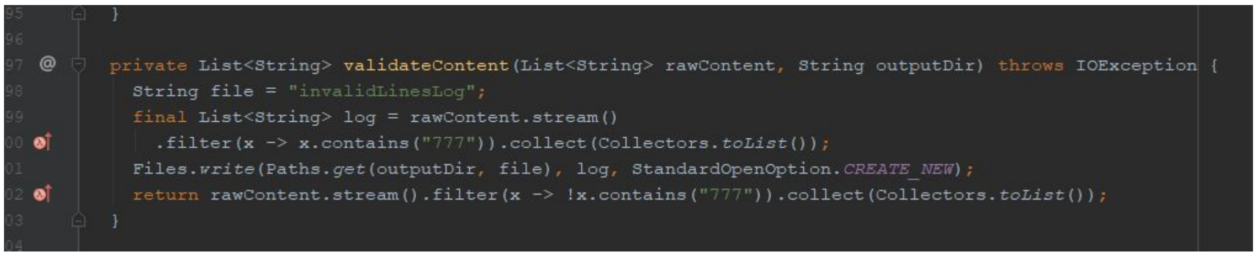
```
29.880386,121.603702,0,0,40228.425625,2010-02-19,10:12:54
29.880371,121.603528,0,0,40228.4256481482,2010-02-19,10:12:56
29.880359,121.603348,0,0,40228.4256712963,2010-02-19,10:12:58
29.880358,121.603166,0,0,40228.4256944444,2010-02-19,10:13:00
29.880357,121.60298,0,0,40228.4257175926,2010-02-19,10:13:02
29.880358,121.602791,0,0,40228.4257407407,2010-02-19,10:13:04
29.880356,121.602581,0,0,40228.4257638889,2010-02-19,10:13:06
29.880349,121.602374,0,0,40228.425787037,2010-02-19,10:13:08
29.880338,121.602148,0,0,40228.4258101852,2010-02-19,10:13:10
29.880332,121.601927,0,0,40228.4258333333,2010-02-19,10:13:12
29.880332,121.601705,0,0,40228.4258564815,2010-02-19,10:13:14
29.880332,121.601489,0,0,40228.4258796296,2010-02-19,10:13:16
29.880333,121.601278,0,0,40228.4259027778,2010-02-19,10:13:18
29.880323,121.601068,0,0,40228.4259259259,2010-02-19,10:13:20
29.880324,121.60085,0,0,40228.4259490741,2010-02-19,10:13:22
29.880329,121.600638,0,0,40228.4259722222,2010-02-19,10:13:24
29.88033,121.600404,0,0,40228.4259953704,2010-02-19,10:13:26
29.880333,121.600161,0,0,40228.4260185185,2010-02-19,10:13:28
29.880334,121.599938,0,0,40228.4260416667,2010-02-19,10:13:30
29.880327,121.599751,0,0,40228.4260648148,2010-02-19,10:13:32
29.880321,121.599604,0,0,40228.426087963,2010-02-19,10:13:34
29.88033,121.599483,0,0,40228.4261111111,2010-02-19,10:13:36
29.880337,121.599369,0,0,40228.4261342593,2010-02-19,10:13:38
29.88034,121.599231,0,0,40228.4261574074,2010-02-19,10:13:40
29.88034,121.599065,0,0,40228.4261805556,2010-02-19,10:13:42
29.880343,121.598872,0,0,40228.4262037037,2010-02-19,10:13:44
29.88035,121.598678,0,0,40228.4262268519,2010-02-19,10:13:46
29.88035,121.598475,0,0,40228.42625,2010-02-19,10:13:48
29.880347,121.598279,0,0,40228.4262731481,2010-02-19,10:13:50
29.88035,121.598074,0,0,40228.4262962963,2010-02-19,10:13:52
29.880343,121.597878,0,0,40228.4263194444,2010-02-19,10:13:54
29.880346,121.597685,0,0,40228.4263425926,2010-02-19,10:13:56
29.880349,121.597475,0,0,40228.4263657407,2010-02-19,10:13:58
29.880347,121.597241,0,0,40228.4263888889,2010-02-19,10:14:00
29.880351,121.596999,0,0,40228.426412037,2010-02-19,10:14:02
```

In order to solve the problem described above, we create the module in our project that will be called GPSDataSimulation which will contain three classes: KafkaMassageProducer, DataSourcesRunner, FileMerger, SeparateDataSourcesSimulator and the first that we will be FileMerger.

FileMerger

class will take several functions:

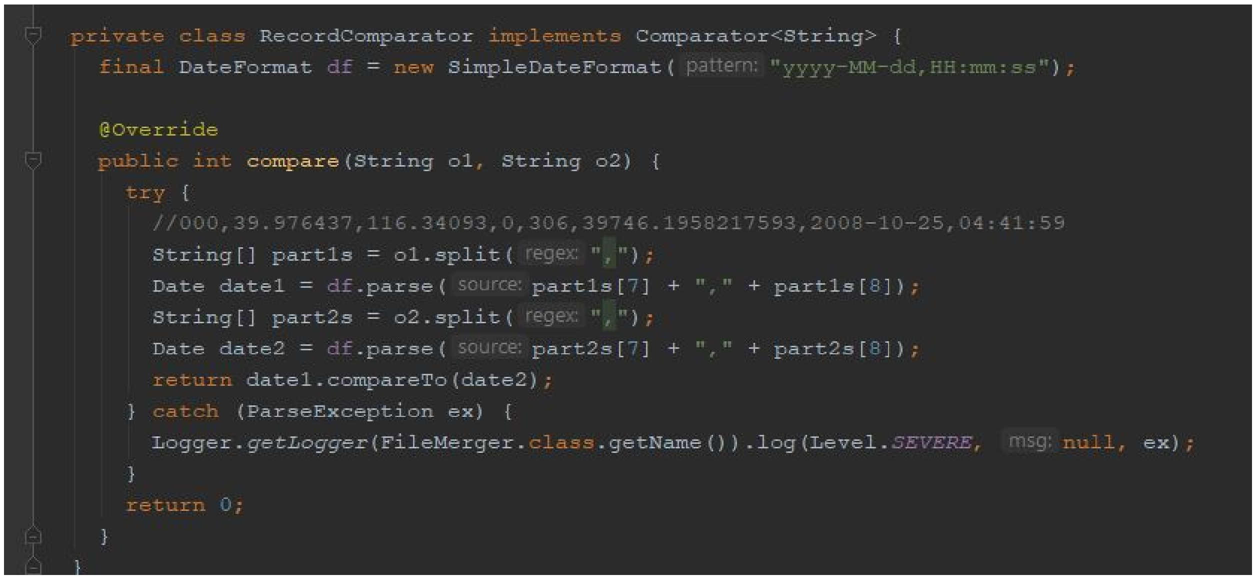
1. read data from all 181 folders
2. Validate the data, filter invalid pieces of data and write it in log file



```java
}

@ private List<String> validateContent(List<String> rawContent, String outputDir) throws IOException {
    String file = "invalidLinesLog";
    final List<String> log = rawContent.stream()
      .filter(x -> x.contains("777")).collect(Collectors.toList());
    Files.write(Paths.get(outputDir, file), log, StandardOpenOption.CREATE_NEW);
    return rawContent.stream().filter(x -> !x.contains("777")).collect(Collectors.toList());
}
```

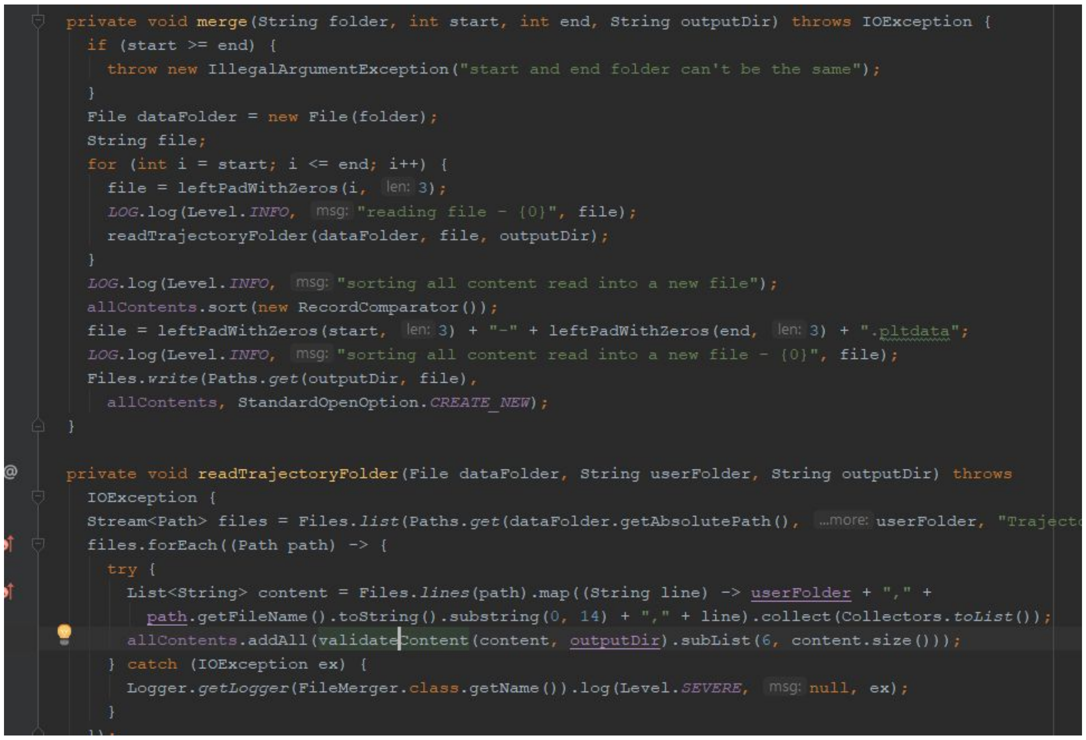
<< Impl of sol 3.0.4 ps >>

3. Sort all data to maintain chronological order



```java
private class RecordComparator implements Comparator<String> {
  final DateFormat df = new SimpleDateFormat( pattern: "yyyy-MM-dd,HH:mm:ss");

  @Override
  public int compare(String o1, String o2) {
    try {
      //000,39.976437,116.34093,0,306,39746.1958217593,2008-10-25,04:41:59
      String[] part1s = o1.split( regex: ",");
      Date date1 = df.parse( source: part1s[7] + "," + part1s[8]);
      String[] part2s = o2.split( regex: ",");
      Date date2 = df.parse( source: part2s[7] + "," + part2s[8]);
      return date1.compareTo(date2);
    } catch (ParseException ex) {
      Logger.getLogger(FileMerger.class.getName()).log(Level.SEVERE,  msg: null, ex);
    }
    return 0;
  }
}
```

4. Merge all 181 with more than 780+ files   into (by default) four separate big files, which will be ready to put in into broker messaging pipeline for further processing and storage



```
private void merge(String folder, int start, int end, String outputDir) throws IOException {
  if (start >= end) {
    throw new IllegalArgumentException("start and end folder can't be the same");
  }
  File dataFolder = new File(folder);
  String file;
  for (int i = start; i <= end; i++) {
    file = leftPadWithZeros(i, len: 3);
    LOG.log(Level.INFO, msg: "reading file - {0}", file);
    readTrajectoryFolder(dataFolder, file, outputDir);
  }
  LOG.log(Level.INFO, msg: "sorting all content read into a new file");
  allContents.sort(new RecordComparator());
  file = leftPadWithZeros(start, len: 3) + "-" + leftPadWithZeros(end, len: 3) + ".pltdata";
  LOG.log(Level.INFO, msg: "sorting all content read into a new file - {0}", file);
  Files.write(Paths.get(outputDir, file),
    allContents, StandardOpenOption.CREATE_NEW);
}

private void readTrajectoryFolder(File dataFolder, String userFolder, String outputDir) throws
  IOException {
  Stream<Path> files = Files.list(Paths.get(dataFolder.getAbsolutePath(), ...more: userFolder, "Traject
  files.forEach((Path path) -> {
    try {
      List<String> content = Files.lines(path).map((String line) -> userFolder + "," +
        path.getFileName().toString().substring(0, 14) + "," + line).collect(Collectors.toList());
      allContents.addAll(validateContent(content, outputDir).subList(6, content.size()));
    } catch (IOException ex) {
      Logger.getLogger(FileMerger.class.getName()).log(Level.SEVERE, msg: null, ex);
    }
```

KafkaMessageProducer

class will be used for :

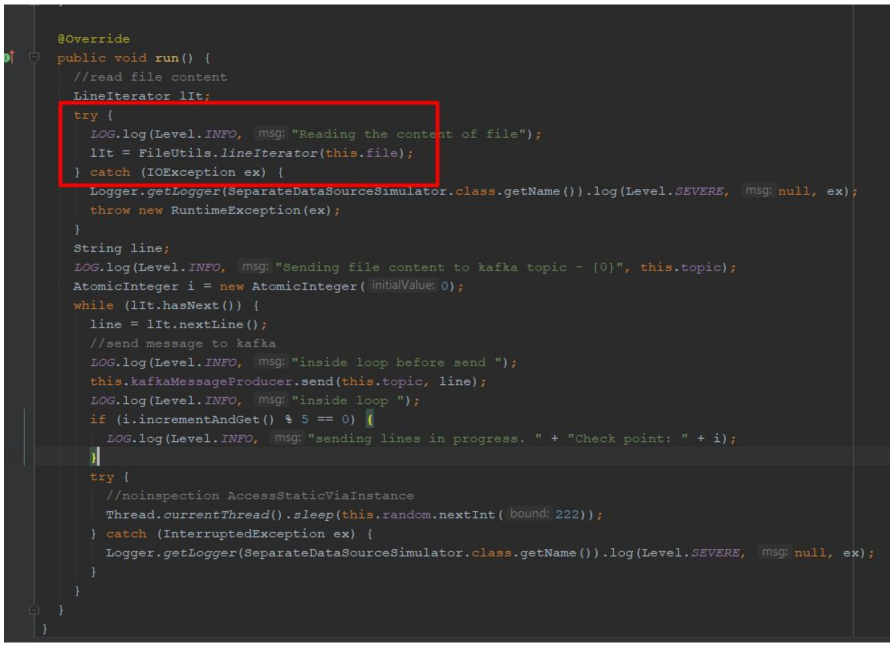
1. Contain all necessary configuration for the message broker system that will be used to consume data for the further processing.

The Apache Kafka broker was chosen because of it guaranty preservation of chronological order of each message delivery which valuable for our use case.

SeparateDataSourceSimulator

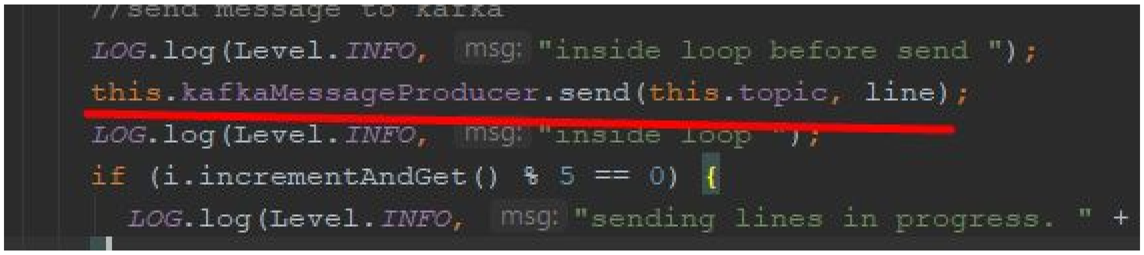
class will be used for :

1. To read each line of the file



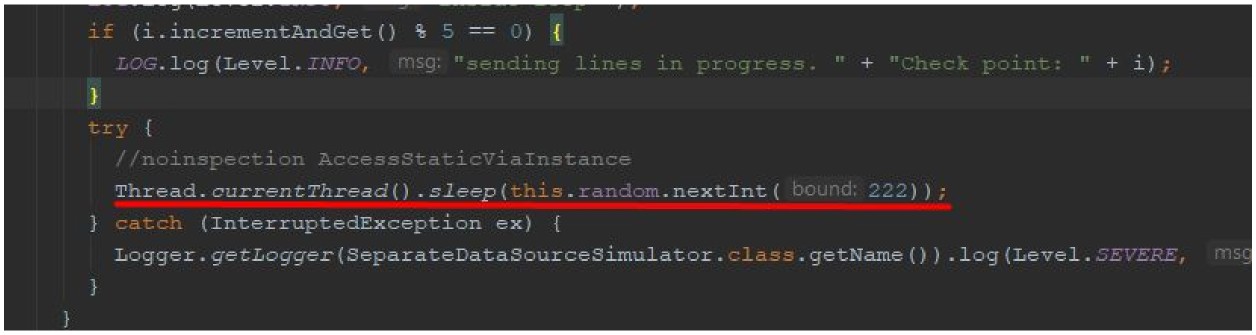
```java
    @Override
    public void run() {
      //read file content
      LineIterator lIt;
      try {
        LOG.log(Level.INFO,  msg: "Reading the content of file");
        lIt = FileUtils.lineIterator(this.file);
      } catch (IOException ex) {
        Logger.getLogger(SeparateDataSourceSimulator.class.getName()).log(Level.SEVERE,  msg: null, ex);
        throw new RuntimeException(ex);
      }
      String line;
      LOG.log(Level.INFO,  msg: "Sending file content to kafka topic - {0}", this.topic);
      AtomicInteger i = new AtomicInteger( initialValue: 0);
      while (lIt.hasNext()) {
        line = lIt.nextLine();
        //send message to kafka
        LOG.log(Level.INFO,  msg: "inside loop before send ");
        this.kafkaMessageProducer.send(this.topic, line);
        LOG.log(Level.INFO,  msg: "inside loop ");
        if (i.incrementAndGet() % 5 == 0) {
          LOG.log(Level.INFO,  msg: "sending lines in progress. " + "Check point: " + i);
        }
        try {
          //noinspection AccessStaticViaInstance
          Thread.currentThread().sleep(this.random.nextInt( bound: 222));
        } catch (InterruptedException ex) {
          Logger.getLogger(SeparateDataSourceSimulator.class.getName()).log(Level.SEVERE,  msg: null, ex);
        }
      }
    }
}
```

2. Produce all necessary logs to control and monitor data flow
3. Write data (line by line) in the Kafka broker



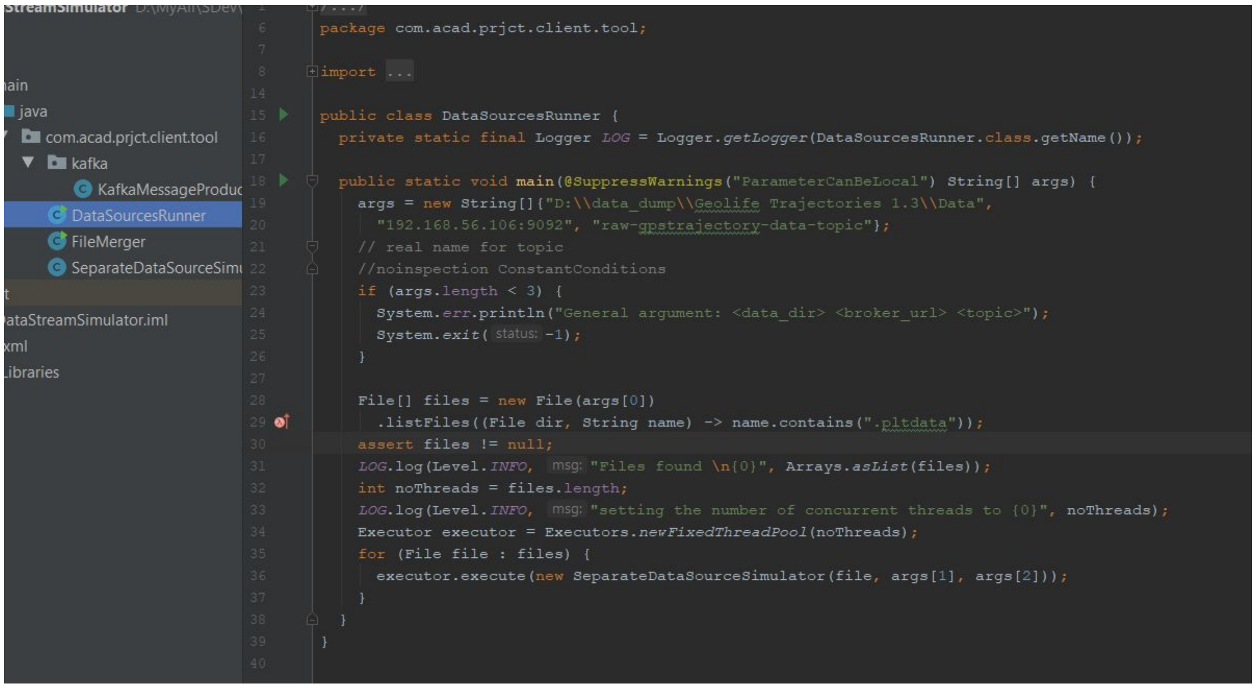
```
//send message to kafka
LOG.log(Level.INFO, msg: "inside loop before send ");
this.kafkaMessageProducer.send(this.topic, line);
LOG.log(Level.INFO, msg: "inside loop ");
if (i.incrementAndGet() % 5 == 0) {
  LOG.log(Level.INFO, msg: "sending lines in progress. " +
```

To avoid CPU overhead we simulate random latency between consuming each piece of data to Kafka broker



```
    if (i.incrementAndGet() % 5 == 0) {
      LOG.log(Level.INFO, msg: "sending lines in progress. " + "Check point: " + i);
    }
    try {
      //noinspection AccessStaticViaInstance
      Thread.currentThread().sleep(this.random.nextInt( bound: 222));
    } catch (InterruptedException ex) {
      Logger.getLogger(SeparateDataSourceSimulator.class.getName()).log(Level.SEVERE, msg
    }
}
```

DataSourcesRunner



```java
package com.acad.prjct.client.tool;

import ...

public class DataSourcesRunner {
  private static final Logger LOG = Logger.getLogger(DataSourcesRunner.class.getName());

  public static void main(@SuppressWarnings("ParameterCanBeLocal") String[] args) {
    args = new String[]{"D:\\data_dump\\Geolife Trajectories 1.3\\Data",
      "192.168.56.106:9092", "raw-gpstrajectory-data-topic"};
    // real name for topic
    //noinspection ConstantConditions
    if (args.length < 3) {
      System.err.println("General argument: <data_dir> <broker_url> <topic>");
      System.exit( status: -1);
    }

    File[] files = new File(args[0])
      .listFiles((File dir, String name) -> name.contains(".pltdata"));
    assert files != null;
    LOG.log(Level.INFO,  msg: "Files found \n{0}", Arrays.asList(files));
    int noThreads = files.length;
    LOG.log(Level.INFO,  msg: "setting the number of concurrent threads to {0}", noThreads);
    Executor executor = Executors.newFixedThreadPool(noThreads);
    for (File file : files) {
      executor.execute(new SeparateDataSourceSimulator(file, args[1], args[2]));
    }
  }
}
```

Finally, the DataSourcesRunner class will be used for :

1. Point to the folder from which the system should take the data and put it in apache Kafka broker
2. Point to the name of the topic that will produce GPS raw data flow after initial validation and preparation
3. Start up several (by default number of threads equals a number of files in the data directory) several threads that will simulate separate data sources that will write to broker messaging system.

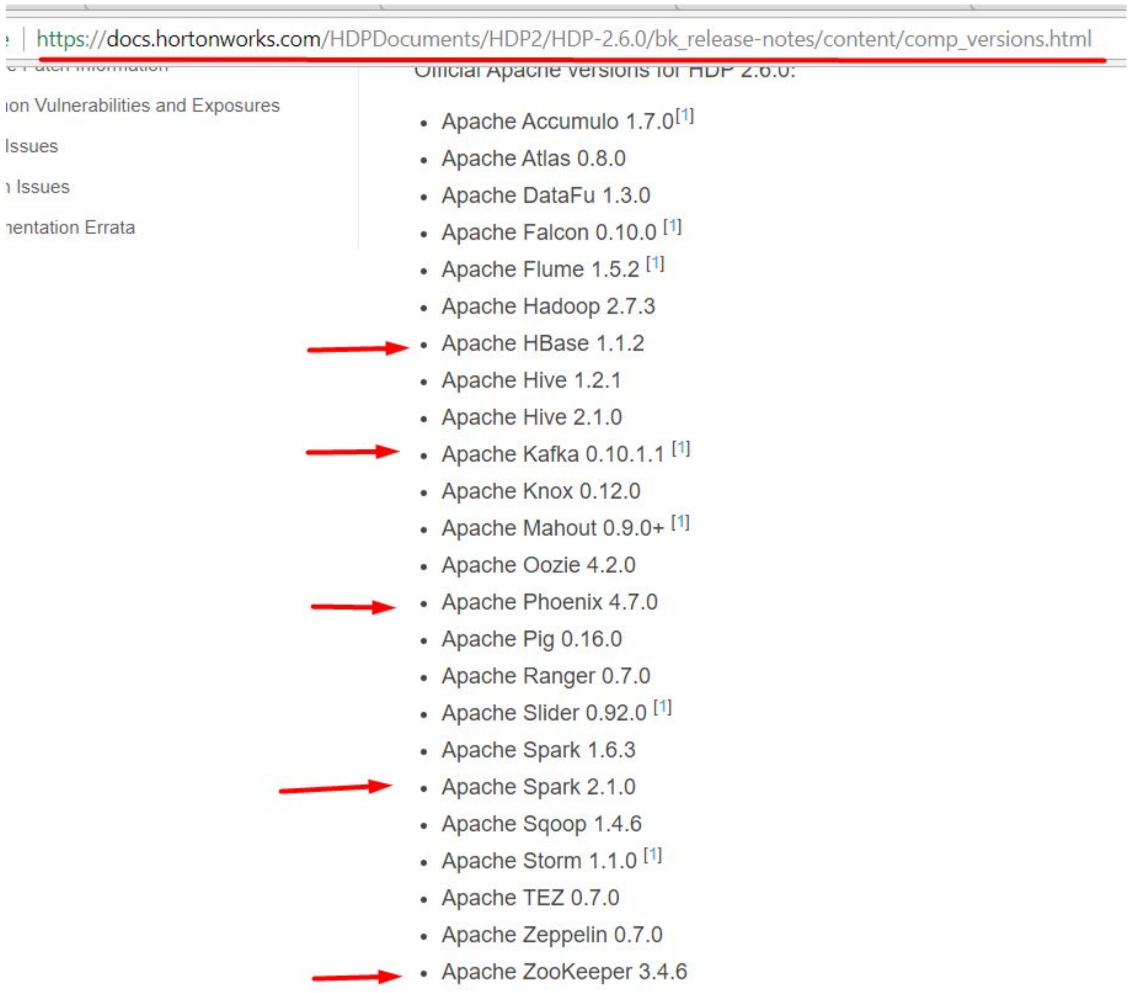
<<impl of solution to 3.0.1 ps>>

<<impl of solution to 3.0.2 ps>>

VM configuration

Choosing component version

In order to meet all necessary architectural requirements and avoid incompatibility of the version of the components let's refer to the industry leaders product and choose the versions of the component for example like in Hortonworks HDP. So in hortonworks.com, we can find all necessary specification of HDP 2.6 and HDP 2.5 and we'll keep in mind all these versions while building our own VM distribution.



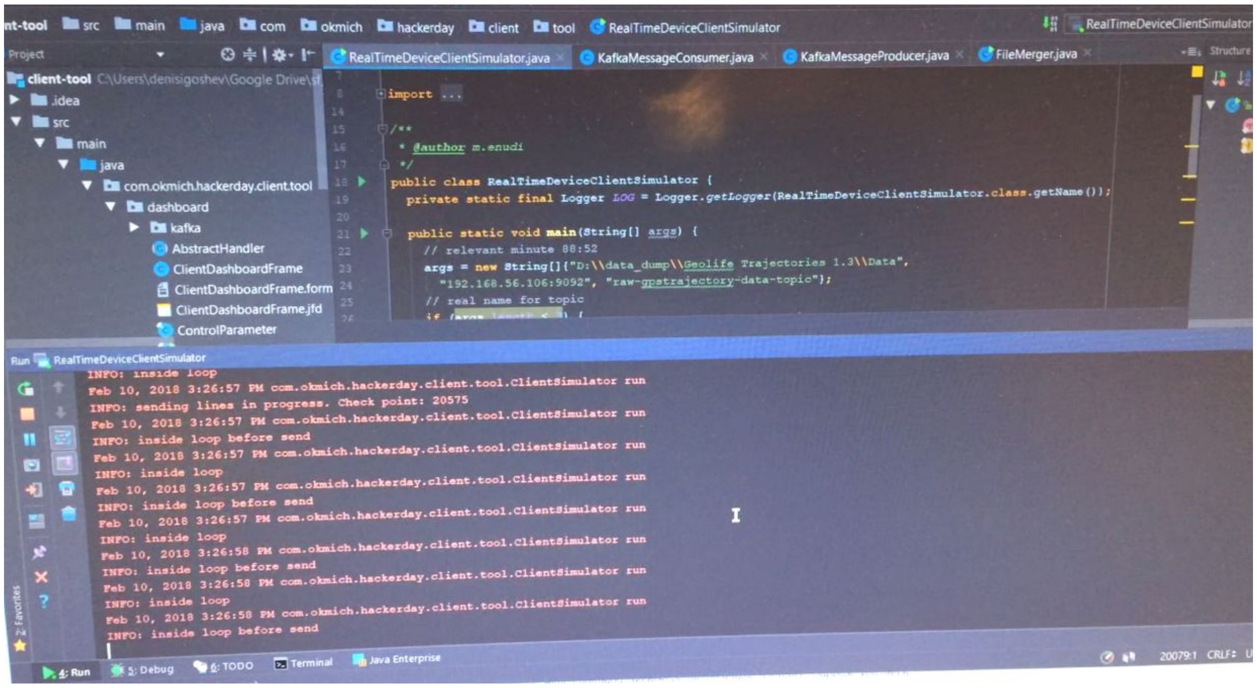
https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.0/bk_release-notes/content/comp_versions.html

Official Apache versions for HDP 2.6.0:

- Apache Accumulo 1.7.0[1]
- Apache Atlas 0.8.0
- Apache DataFu 1.3.0
- Apache Falcon 0.10.0 [1]
- Apache Flume 1.5.2 [1]
- Apache Hadoop 2.7.3
- Apache HBase 1.1.2
- Apache Hive 1.2.1
- Apache Hive 2.1.0
- Apache Kafka 0.10.1.1 [1]
- Apache Knox 0.12.0
- Apache Mahout 0.9.0+ [1]
- Apache Oozie 4.2.0
- Apache Phoenix 4.7.0
- Apache Pig 0.16.0
- Apache Ranger 0.7.0
- Apache Slider 0.92.0 [1]
- Apache Spark 1.6.3
- Apache Spark 2.1.0
- Apache Sqoop 1.4.6
- Apache Storm 1.1.0 [1]
- Apache TEZ 0.7.0
- Apache Zeppelin 0.7.0
- Apache ZooKeeper 3.4.6

Push data stream to Apache Kafka

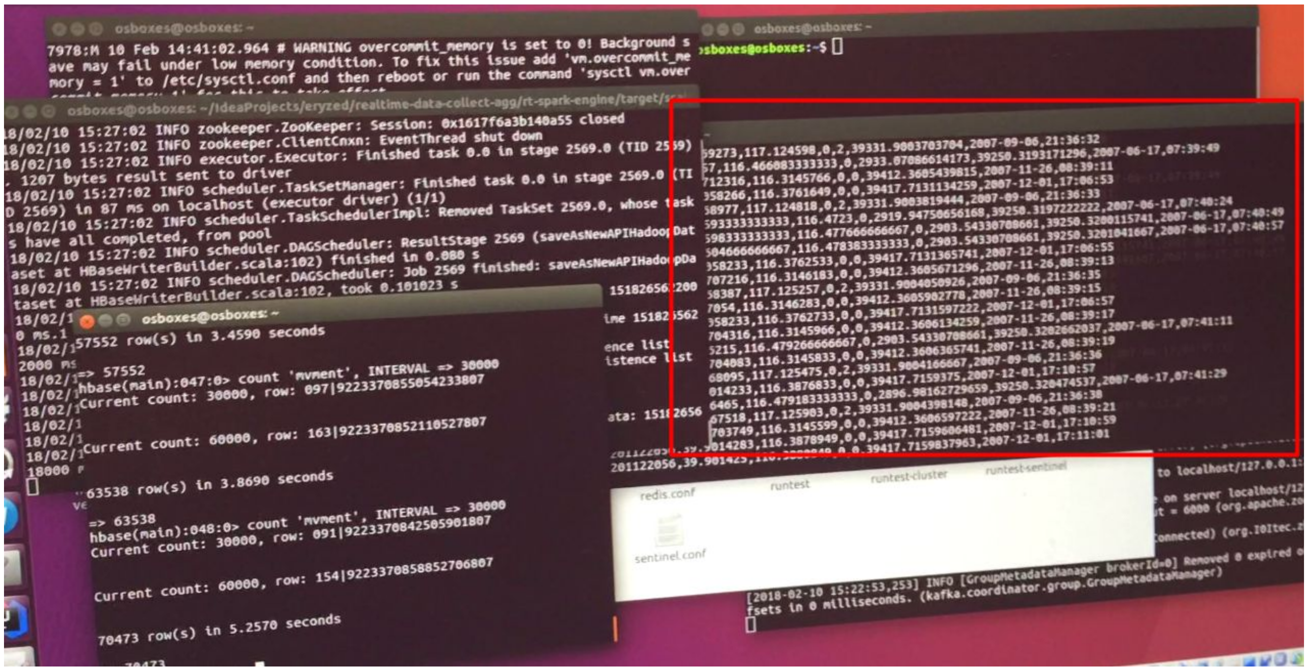
In order to check that the data successfully could put and retrieve from our messaging broker, we can create console consumer in VM ubuntu OS and take a look does data appears in terminal window or not



```java
import ...

/**
 * @author m.enudi
 */
public class RealTimeDeviceClientSimulator {
  private static final Logger LOG = Logger.getLogger(RealTimeDeviceClientSimulator.class.getName());

  public static void main(String[] args) {
    // relevant minute 88:52
    args = new String[]{"D:\\data_dump\\Geolife Trajectories 1.3\\Data",
      "192.168.56.106:9092", "raw-gpstrajectory-data-topic"};
    // real name for topic
```

```
INFO: inside loop
Feb 10, 2018 3:26:57 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: sending lines in progress. Check point: 20575
Feb 10, 2018 3:26:57 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop before send
Feb 10, 2018 3:26:57 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop
Feb 10, 2018 3:26:57 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop before send
Feb 10, 2018 3:26:57 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop
Feb 10, 2018 3:26:58 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop before send
Feb 10, 2018 3:26:58 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop
Feb 10, 2018 3:26:58 PM com.okmich.hackerday.client.tool.ClientSimulator run
INFO: inside loop before send
```

Where the data that is pushing to message broker



```
7978:M 10 Feb 14:41:02.964 # WARNING overcommit_memory is set to 0! Background s
ave may fail under low memory condition. To fix this issue add 'vm.overcommit_me
mory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.over
```

```
osboxes@osboxes: ~/IdeaProjects/eryzed/realtime-data-collect-agg/rt-spark-engine/target/
18/02/10 15:27:02 INFO zookeeper.ZooKeeper: Session: 0x1617f6a3b140a55 closed
18/02/10 15:27:02 INFO zookeeper.ClientCnxn: EventThread shut down
18/02/10 15:27:02 INFO executor.Executor: Finished task 0.0 in stage 2569.0 (TID 2569)
. 1207 bytes result sent to driver
18/02/10 15:27:02 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 2569.0 (TI
D 2569) in 87 ms on localhost (executor driver) (1/1)
18/02/10 15:27:02 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2569.0, whose task
s have all completed, from pool
18/02/10 15:27:02 INFO scheduler.DAGScheduler: ResultStage 2569 (saveAsNewAPIHadoopDat
aset at HBaseWriterBuilder.scala:102) finished in 0.080 s
18/02/10 15:27:02 INFO scheduler.DAGScheduler: Job 2569 finished: saveAsNewAPIHadoopDa
taset at HBaseWriterBuilder.scala:102, took 0.101023 s
```

```
osboxes@osboxes: ~
57552 row(s) in 3.4590 seconds

=> 57552
hbase(main):047:0> count 'mvment', INTERVAL => 30000
Current count: 30000, row: 097|9223370855054233807

Current count: 60000, row: 163|9223370852110527807

63538 row(s) in 3.8690 seconds

=> 63538
hbase(main):048:0> count 'mvment', INTERVAL => 30000
Current count: 30000, row: 091|9223370842505901807

Current count: 60000, row: 154|9223370858852706807

70473 row(s) in 5.2570 seconds
```

```
osboxes@osboxes:~$
```

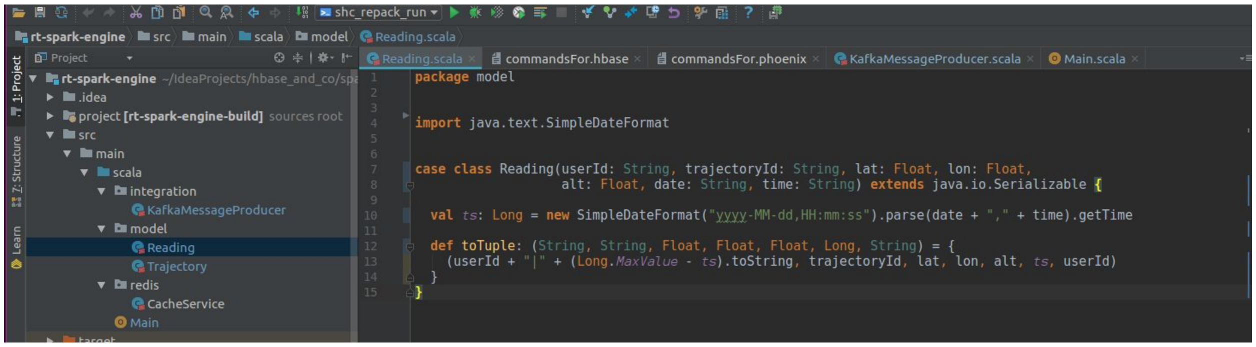
```
59273,117.124598,0,2,39331.9003703704,2007-09-06,21:36:32
57,116.466083333333,0,2933.07086614173,39250.3193171296,2007-06-17,07:39:49
712316,116.3145766,0,0,39412.3605439815,2007-11-26,08:39:11
958266,116.3761649,0,0,39417.7131134259,2007-12-01,17:06:53
58977,117.124818,0,2,39331.9003819444,2007-09-06,21:36:33
593333333333,116.4723,0,2919.94750656168,39250.3197222222,2007-06-17,07:40:24
598333333333,116.477666666667,0,2903.54330708661,39250.3200115741,2007-06-17,07:40:49
504666666667,116.478383333333,0,2903.54330708661,39250.3201041667,2007-06-17,07:40:57
958233,116.3762533,0,0,39417.7131365741,2007-12-01,17:06:55
707216,116.3146183,0,0,39412.3605671296,2007-11-26,08:39:13
58387,117.125257,0,2,39331.9004050926,2007-09-06,21:36:35
7054,116.3146283,0,0,39412.3605902778,2007-11-26,08:39:15
958233,116.3762733,0,0,39417.7131597222,2007-12-01,17:06:57
704316,116.3145966,0,0,39412.3606134259,2007-11-26,08:39:17
5215,116.479266666667,0,2903.54330708661,39250.3202662037,2007-06-17,07:41:11
704083,116.3145833,0,0,39412.3606365741,2007-11-26,08:39:19
68095,117.125475,0,2,39331.9004166667,2007-09-06,21:36:36
014233,116.3876833,0,0,39417.7159375,2007-12-01,17:10:57
6465,116.479183333333,0,2896.98162729659,39250.320474537,2007-06-17,07:41:29
67518,117.125903,0,2,39331.9004398148,2007-09-06,21:36:38
703749,116.3145599,0,0,39412.3606597222,2007-11-26,08:39:21
014283,116.3878949,0,0,39417.7159606481,2007-12-01,17:10:59
```

Here the same data as u can see but on the other side of the broker (push out to the console window)
Now we can move to another step is that pushing data stream to HBase.

4. Data Enrichment

Push data stream to HDFS Database

1. We need to create tables in HBase
    a. For this purpose, we will use HBase shell
    b. We need to create ‘mvment’ table with column family ‘main’ - for the raw valid data (to satisfy problem statement requirements << *** type the requirement point >>
    c. We create ‘mvment_by_traj’ table with column family ‘main’ in which enrichment data will be collected and against which we will make our analysis (from point 5 the case problem statement)
2. Now we need to push out data from broker message to apache spark streaming
3. In Apache Spark (rt-spark.jar) we will make :
    a. Further validation and enrichment todo >> add name of class
    b. Push data stream (one micro batch at the single iteration) to HBase todo >> add name of class



```scala
package model

import java.text.SimpleDateFormat

case class Reading(userId: String, trajectoryId: String, lat: Float, lon: Float,
                   alt: Float, date: String, time: String) extends java.io.Serializable {

  val ts: Long = new SimpleDateFormat("yyyy-MM-dd,HH:mm:ss").parse(date + "," + time).getTime

  def toTuple: (String, String, Float, Float, Float, Long, String) = {
    (userId + "|" + (Long.MaxValue - ts).toString, trajectoryId, lat, lon, alt, ts, userId)
  }
}
```

<< Impl of sol 3.0.3 ps >>

<< Impl of sol 3.0.5 ps >>

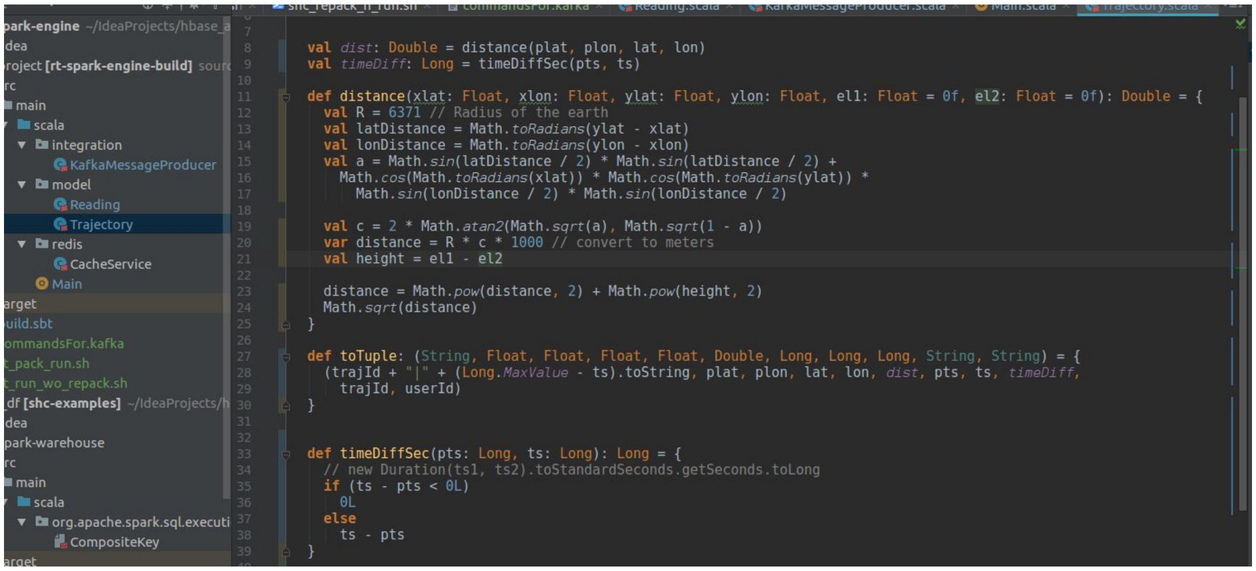
<< Impl of sol 4.1.1 ps >>

<< Impl of sol 4.1.2 ps >>

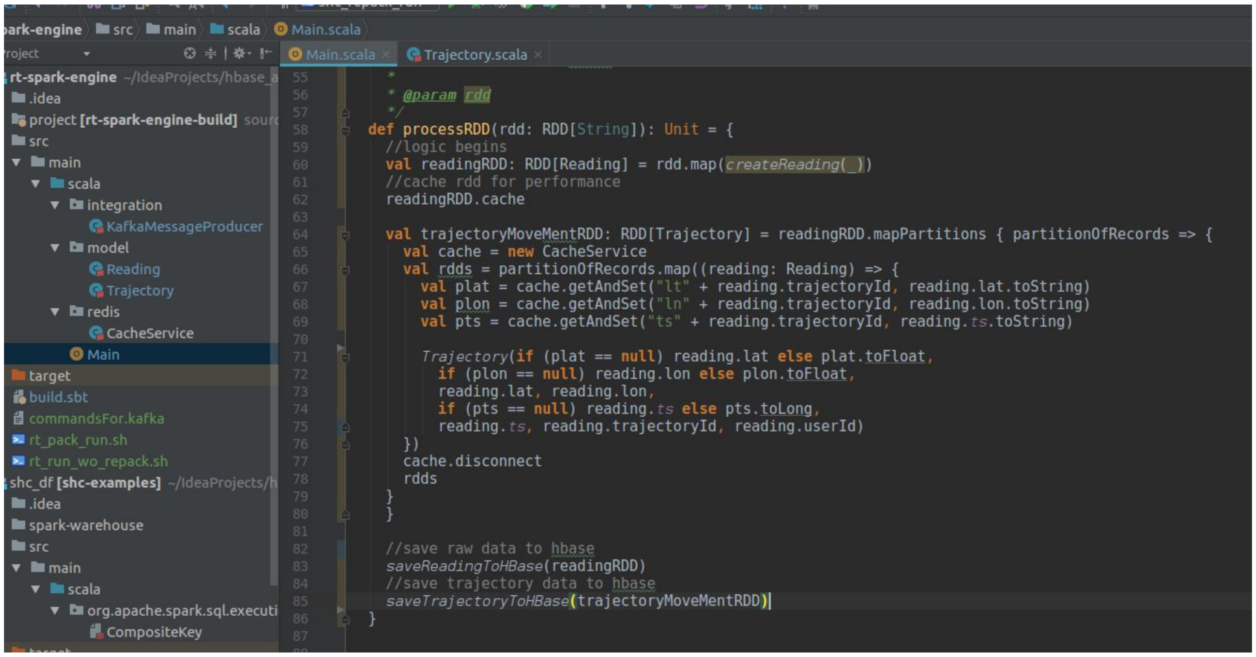
<< Impl of sol 4.1.3 ps >>

<< Impl of sol 5.1.2 ps >>

Because trajectory model contain “userId” field and all necessary fields for making advanced analytics of data database by could completely avoid usage of such expensive operation as a SQL query << JOIN>>



```scala
  val dist: Double = distance(plat, plon, lat, lon)
  val timeDiff: Long = timeDiffSec(pts, ts)

  def distance(xlat: Float, xlon: Float, ylat: Float, ylon: Float, el1: Float = 0f, el2: Float = 0f): Double = {
    val R = 6371 // Radius of the earth
    val latDistance = Math.toRadians(ylat - xlat)
    val lonDistance = Math.toRadians(ylon - xlon)
    val a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2) +
      Math.cos(Math.toRadians(xlat)) * Math.cos(Math.toRadians(ylat)) *
        Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2)

    val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
    var distance = R * c * 1000 // convert to meters
    val height = el1 - el2

    distance = Math.pow(distance, 2) + Math.pow(height, 2)
    Math.sqrt(distance)
  }

  def toTuple: (String, Float, Float, Float, Float, Double, Long, Long, Long, String, String) = {
    (trajId + "|" + (Long.MaxValue - ts).toString, plat, plon, lat, lon, dist, pts, ts, timeDiff,
      trajId, userId)
  }


  def timeDiffSec(pts: Long, ts: Long): Long = {
    // new Duration(ts1, ts2).toStandardSeconds.getSeconds.toLong
    if (ts - pts < 0L)
      0L
    else
      ts - pts
  }
```



```scala
  *
  * @param rdd
  */
def processRDD(rdd: RDD[String]): Unit = {
  //logic begins
  val readingRDD: RDD[Reading] = rdd.map(createReading(_))
  //cache rdd for performance
  readingRDD.cache

  val trajectoryMoveMentRDD: RDD[Trajectory] = readingRDD.mapPartitions { partitionOfRecords => {
    val cache = new CacheService
    val rdds = partitionOfRecords.map((reading: Reading) => {
      val plat = cache.getAndSet("lt" + reading.trajectoryId, reading.lat.toString)
      val plon = cache.getAndSet("ln" + reading.trajectoryId, reading.lon.toString)
      val pts = cache.getAndSet("ts" + reading.trajectoryId, reading.ts.toString)

      Trajectory(if (plat == null) reading.lat else plat.toFloat,
        if (plon == null) reading.lon else plon.toFloat,
        reading.lat, reading.lon,
        if (pts == null) reading.ts else pts.toLong,
        reading.ts, reading.trajectoryId, reading.userId)
    })
    cache.disconnect
    rdds
  }
  }

  //save raw data to hbase
  saveReadingToHBase(readingRDD)
  //save trajectory data to hbase
  saveTrajectoryToHBase(trajectoryMoveMentRDD)
}
```

<< Impl of sol 4.1.4 ps >>

<< Impl of sol 4.2.1 ps >>

<< Impl of sol 4.2.2 ps >>

<< Impl of sol 5.1.1 ps >>

<< Impl of sol 5.1.5 ps >>

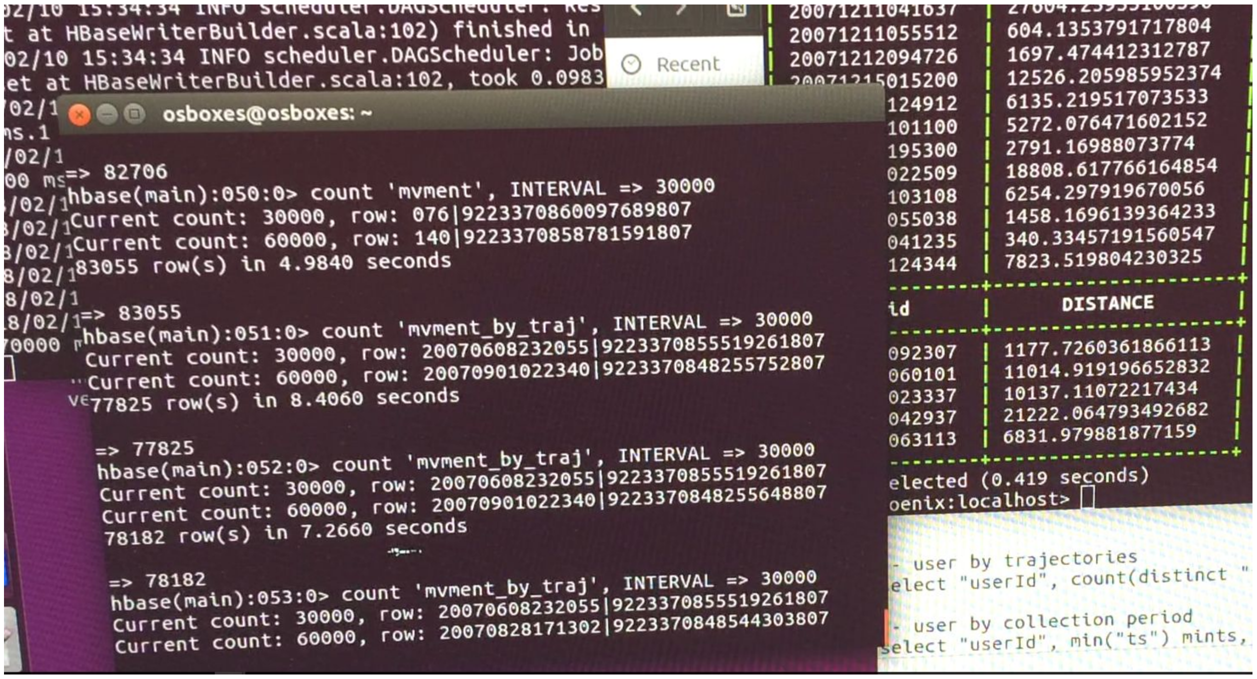
Data flow monitoring and initial analysis

If we use just four threads to simulated 4 difference GPS devices with random latency (0 - 200 ms) the transfer of 2 Gb dataset that initially was given to us to transfer to hdfs and substantially analyze will take a lot of time.

In order to make sure that during this process the data indeed comes to HBase, we need to adopt at least a simple monitor tools that alert as in case the system failure. For this purpose, we will be used :

1. Hbase shell
2. Apache Phoenix

So this two tools will show us that indeed the amount of data in HBase is increasing during the transfer and validation is apache spark stream going well



```
t at HBaseWriterBuilder.scala:102) finished in
02/10 15:34:34 INFO scheduler.DAGScheduler: Job
et at HBaseWriterBuilder.scala:102, took 0.0983
```

```
osboxes@osboxes: ~

=> 82706
hbase(main):050:0> count 'mvment', INTERVAL => 30000
Current count: 30000, row: 076|9223370860097689807
Current count: 60000, row: 140|9223370858781591807
83055 row(s) in 4.9840 seconds

=> 83055
hbase(main):051:0> count 'mvment_by_traj', INTERVAL => 30000
Current count: 30000, row: 20070608232055|9223370855519261807
Current count: 60000, row: 20070901022340|9223370848255752807
77825 row(s) in 8.4060 seconds

=> 77825
hbase(main):052:0> count 'mvment_by_traj', INTERVAL => 30000
Current count: 30000, row: 20070608232055|9223370855519261807
Current count: 60000, row: 20070901022340|9223370848255648807
78182 row(s) in 7.2660 seconds

=> 78182
hbase(main):053:0> count 'mvment_by_traj', INTERVAL => 30000
Current count: 30000, row: 20070608232055|9223370855519261807
Current count: 60000, row: 20070828171302|9223370848544303807
```

```
| 20071211055512 | 604.1353791717804  |
| 20071212094726 | 1697.474412312787  |
| 20071215015200 | 12526.205985952374 |
          124912 | 6135.219517073533  |
          101100 | 5272.076471602152  |
          195300 | 2791.16988073774   |
          022509 | 18808.617766164854 |
          103108 | 6254.297919670056  |
          055038 | 1458.1696139364233 |
          041235 | 340.33457191560547 |
          124344 | 7823.519804230325  |
---------+--------------------+
id       |      DISTANCE      |
---------+--------------------+
092307   | 1177.7260361866113 |
060101   | 11014.919196652832 |
023337   | 10137.11072217434  |
042937   | 21222.064793492682 |
063113   | 6831.979881877159  |
---------+--------------------+
elected (0.419 seconds)
oenix:localhost>
```

```
- user by trajectories
elect "userId", count(distinct "
- user by collection period
select "userId", min("ts") mints,
```

<< Impl of sol 5.1.3 ps >>

5. Data analysis

Now will query the data by using apache spark sql that connects to HBase through hortonworks connector for building further more advanced analysis of the data
To check that indeed all connectivity between apache spark SQL and HBase going well we can run the same query in apache phoenix. In case of receiving the different result from the same query, it will alert us about some system malfunction. So good to have apache phoenix shell or squirrel GUI ready for work
Because we use HortonWorks hbase - spark connector it enables to run the apache spark job on the top of the YARN which is important as a part of the assignment 5.1.4

<< Impl of sol 5.1.4 ps >>

<< Impl of sol 5.0 ps >>

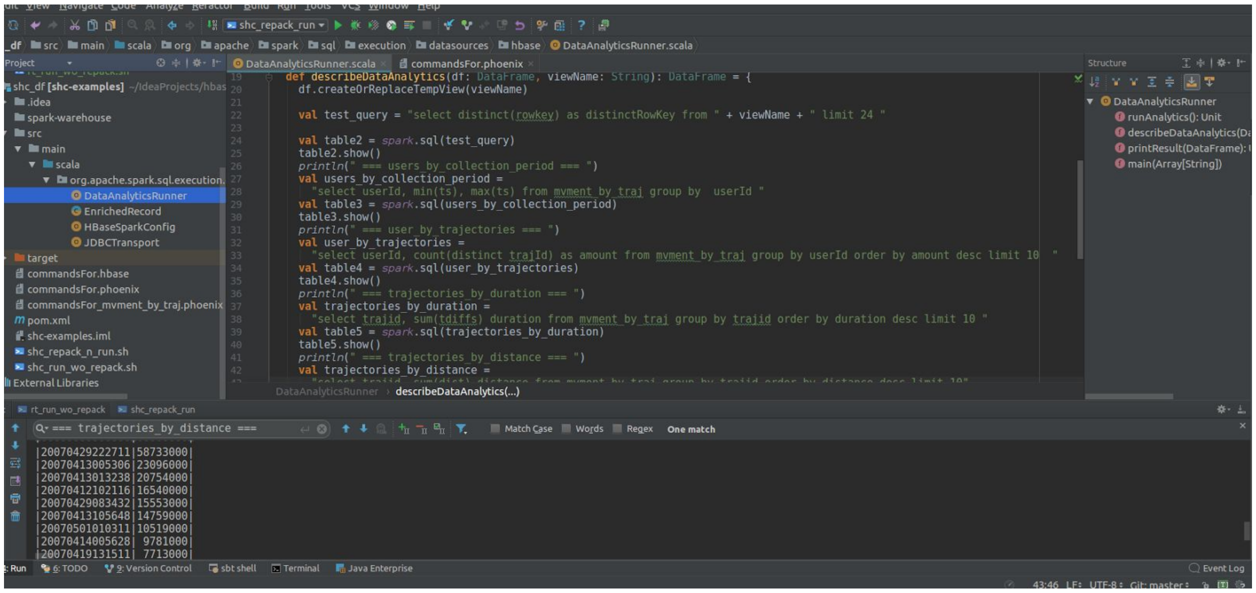
<< Impl of sol 5.0.1 ps >>

<< Impl of sol 5.0.2 ps >>

<< Impl of sol 5.0.3 ps >>

<< Impl of sol 5.0.4 ps >>

<< Impl of sol 5.0.5 ps >>

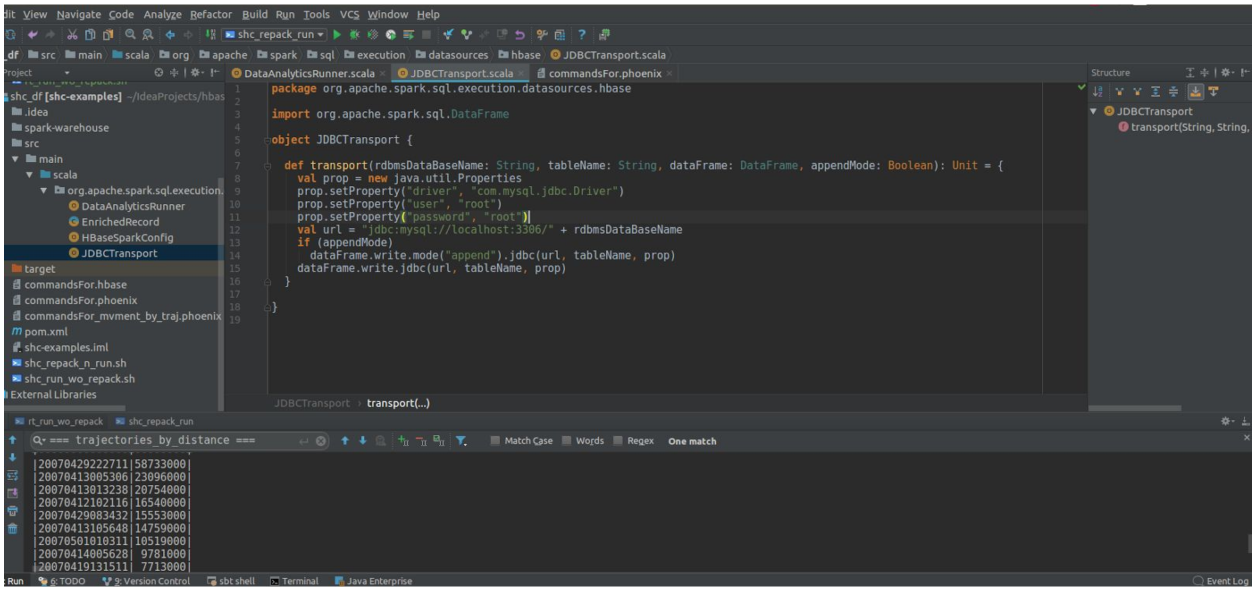


```scala
  def describeDataAnalytics(df: DataFrame, viewName: String): DataFrame = {
    df.createOrReplaceTempView(viewName)

    val test_query = "select distinct(rowkey) as distinctRowKey from " + viewName + " limit 24 "

    val table2 = spark.sql(test_query)
    table2.show()
    println(" === users_by_collection_period === ")
    val users_by_collection_period =
      "select userId, min(ts), max(ts) from mvment_by_traj group by  userId "
    val table3 = spark.sql(users_by_collection_period)
    table3.show()
    println(" === user_by_trajectories === ")
    val user_by_trajectories =
      "select userId, count(distinct trajId) as amount from mvment_by_traj group by userId order by amount desc limit 10  "
    val table4 = spark.sql(user_by_trajectories)
    table4.show()
    println(" === trajectories_by_duration === ")
    val trajectories_by_duration =
      "select trajid, sum(tdiffs) duration from mvment_by_traj group by trajid order by duration desc limit 10 "
    val table5 = spark.sql(trajectories_by_duration)
    table5.show()
    println(" === trajectories_by_distance === ")
    val trajectories_by_distance =
```

```
|20070429222711|58733000|
|20070413005306|23096000|
|20070413013238|20754000|
|20070412102116|16540000|
|20070429083432|15553000|
|20070413105648|14759000|
|20070501010311|10519000|
|20070414005628| 9781000|
|20070419131511| 7713000|
```

6. Post Analysis

Transfer and save analytical result to RDBMS database



```scala
package org.apache.spark.sql.execution.datasources.hbase

import org.apache.spark.sql.DataFrame

object JDBCTransport {

  def transport(rdbmsDataBaseName: String, tableName: String, dataFrame: DataFrame, appendMode: Boolean): Unit = {
    val prop = new java.util.Properties
    prop.setProperty("driver", "com.mysql.jdbc.Driver")
    prop.setProperty("user", "root")
    prop.setProperty("password", "root")
    val url = "jdbc:mysql://localhost:3306/" + rdbmsDataBaseName
    if (appendMode)
      dataFrame.write.mode("append").jdbc(url, tableName, prop)
    dataFrame.write.jdbc(url, tableName, prop)
  }

}
```

```
=== trajectories_by_distance ===
|20070429222711|58733000|
|20070413005306|23096000|
|20070413013238|20754000|
|20070412102116|16540000|
|20070429083432|15553000|
|20070413105648|14759000|
|20070501010311|10519000|
|20070414005628| 9781000|
|20070419131511| 7713000|
```

<< Impl of sol 6.0.1 ps >>

Appendix :

Problems outline

3. Data Ingestion and Initial Validation

✓ 3.0.1 Data comes as a stream real time in several data sources simultaneously

✓ 3.0.2 All the timestamp fields in data coming from data sources are of the format YYYY-MM-DD HH:MM:SS.

✓ 3.0.3 Finally, all timestamps must have the format of a long integer to be interpreted as UNIX timestamps when they reach database in HDFS.

✓ 3.0.4 Altitude in feet (-777 if not valid).

✓ 3.0.5 Create an identifier for each data line.

4. Data Enrichment

4.1 Rules for data enrichment

✓ 4.1.1 to each log line should be added “dist” column which represents the geo trajectory distance between current log line and previous chronological log line for the corresponding trajectory

✓ 4.1.2 to each log line should be calculated “tdiffs” column which represents the time difference between current log line and previous chronological log line for the corresponding trajectory

✓ 4.1.3 if the previous timestamp of corresponding trajectory not older than current timestamp of the trajectory set the time difference is 0

✓ 4.1.4 each calculation should be performed in real-time or near real-time mode

4.2 Post Enrichment

✓ 4.2.1 Maintain a copy of valid raw GPS records in HBase.

✓ 4.2.2 Move all valid records in HBase

5. Data analysis

☑ 5.0 To design the system that will be able to make the next analytics :

☑ 5.0.1 Show users and their data collection period

☑ 5.0.2 Determine top 10 users amount of trajectories.

☑ 5.0.3 Determine top 10 trajectories by duration.

☑ 5.0.4 Determine top 10 endured trajectory and which user they belong

5.1. Challenges and optimization

☑ 5.1.1 Raw data and processed data comes to NoSQL database as a stream. All calculations and validations performed as a data streams.

☑ 5.1.2 Try to make joins as less expensive as possible.

☑ 5.1.3 Adopt appropriate monitoring to maintained to track the behavior and overcome failures in the pipeline.

☑ 5.1.4 For data analytics in spark use technologies that could be work with Hadoop resource manager like YARN.

✓ 5.1.5 Write data to HBase as a stream.

6. Post Analysis

✓ 6.0.1. Design functionally to enable result of analytics move to the RDBMS for data storage and quick retrieval.

Appendix 2 :

Video demos :

https://streamable.com/7axck - running_sql_pnoenix_spark
https://streamable.com/jqdt3 - demo_tranfer_from_win_to_ubuntu_via_kafka
https://streamable.com/ks5ru - 50k_lines_of_data_monitoring_pnoenix_hbase_shell
https://streamable.com/wiaqa - 100k_lines_of_data_monitoring_pnoenix_hbase_shell
https://streamable.com/p3xoi - 150k_lines_of_data_monitoring_pnoenix_hbase_shell
https://streamable.com/d9ebv - 200k_lines_of_data_monitoring_pnoenix_hbase_shell
https://streamable.com/6n6o4 - 300k_lines_of_data_monitoring_pnoenix_hbase_shell