

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.

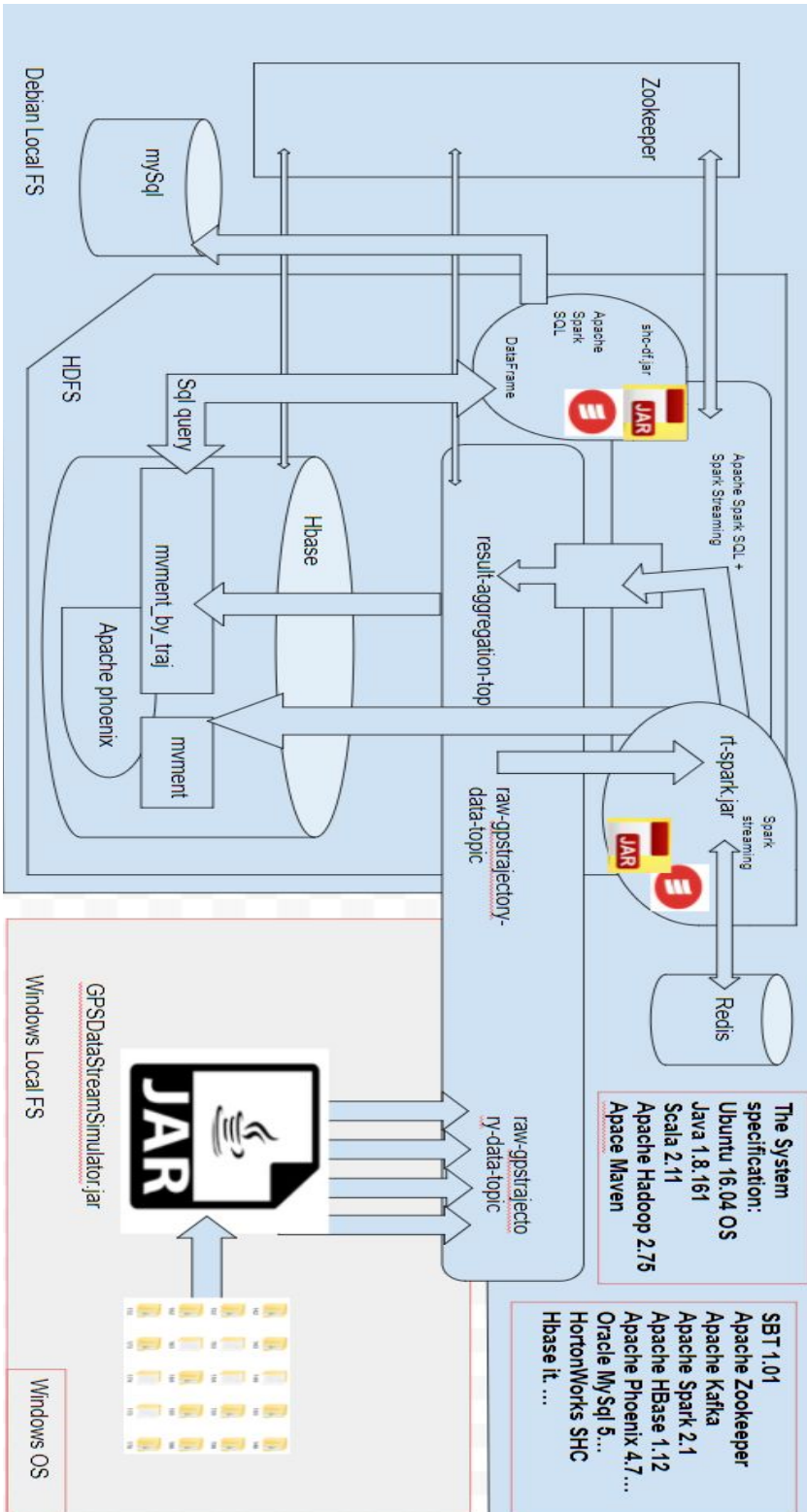
At the end of this document, you can find the hyperlink to the video demos of data flow and jar output that prove in some degree that indeed the codebase can produce the needed result without bugs or exceptions.

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.

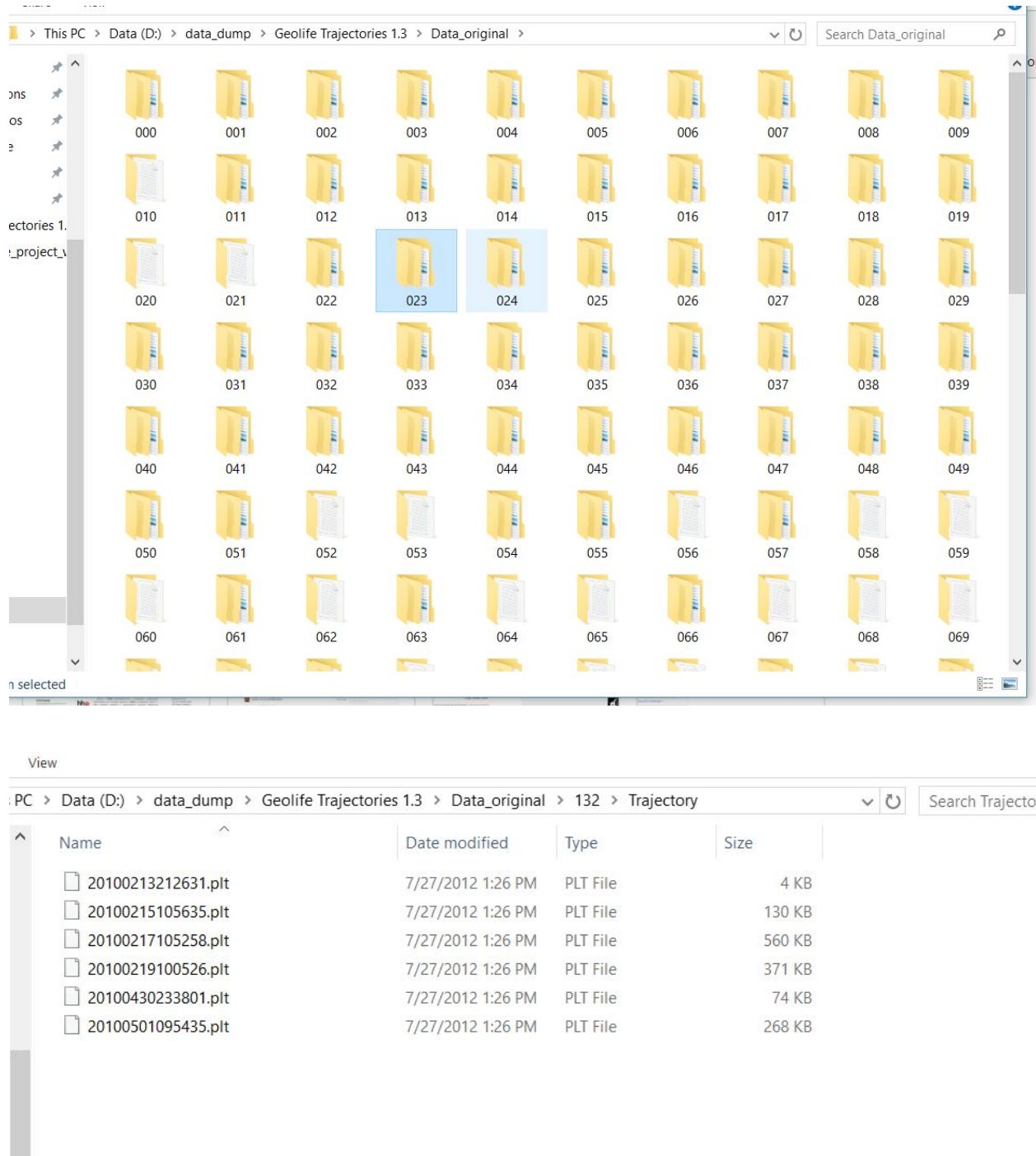
If you will need some elaboration u can always contact me for further comments or consultations.

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

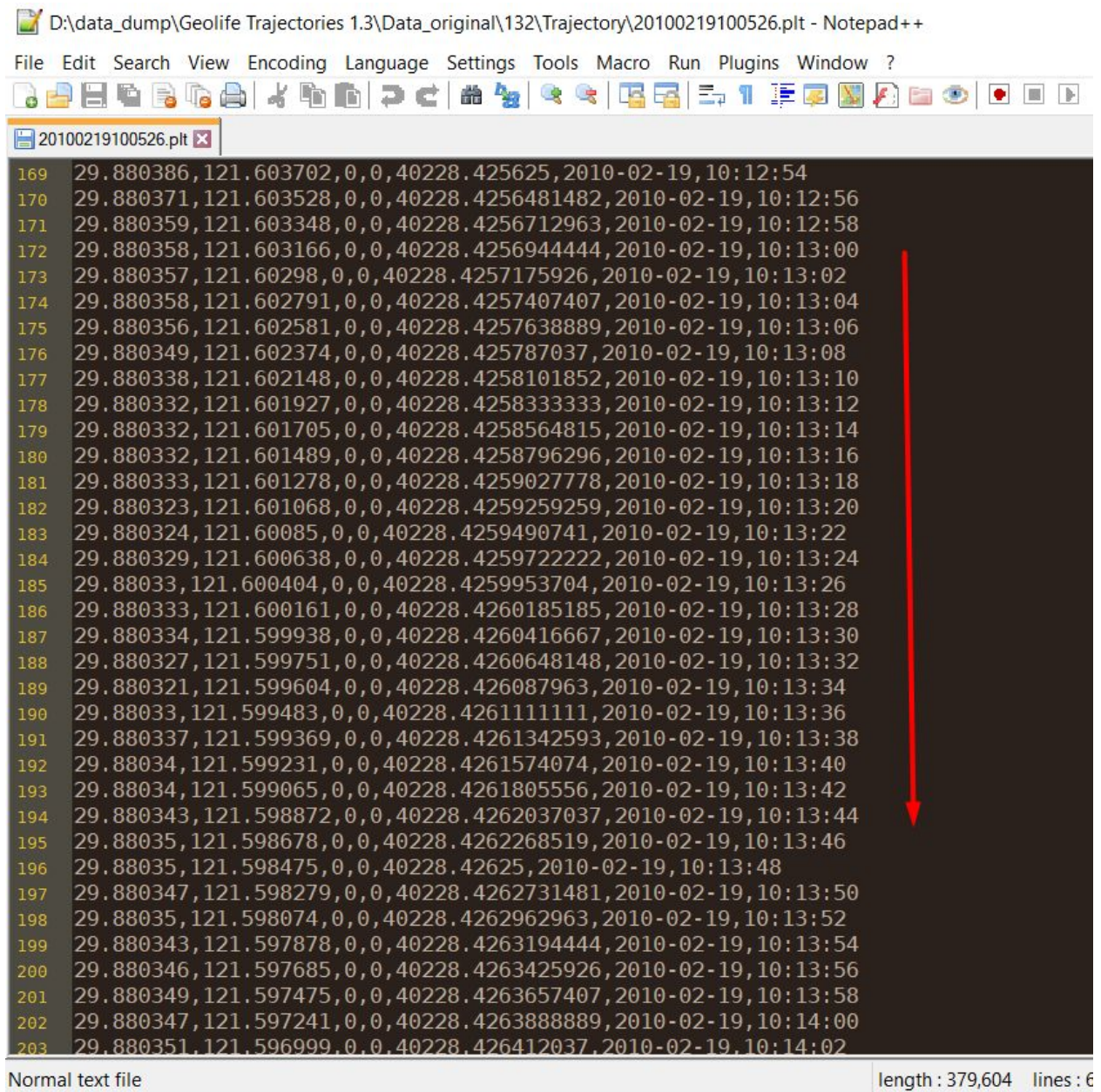


The screenshot displays a Windows File Explorer window. The address bar shows the path: This PC > Data (D:) > data_dump > Geolife Trajectories 1.3 > Data_original >. The main area shows a grid of folders numbered 000 to 069. Folders 023 and 024 are highlighted with a blue selection box. Below the grid, a 'View' section shows a detailed list of files for the selected folder (132 > Trajectory).

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
169 29.880386,121.603702,0,0,40228.425625,2010-02-19,10:12:54
170 29.880371,121.603528,0,0,40228.4256481482,2010-02-19,10:12:56
171 29.880359,121.603348,0,0,40228.4256712963,2010-02-19,10:12:58
172 29.880358,121.603166,0,0,40228.4256944444,2010-02-19,10:13:00
173 29.880357,121.60298,0,0,40228.4257175926,2010-02-19,10:13:02
174 29.880358,121.602791,0,0,40228.4257407407,2010-02-19,10:13:04
175 29.880356,121.602581,0,0,40228.4257638889,2010-02-19,10:13:06
176 29.880349,121.602374,0,0,40228.425787037,2010-02-19,10:13:08
177 29.880338,121.602148,0,0,40228.4258101852,2010-02-19,10:13:10
178 29.880332,121.601927,0,0,40228.4258333333,2010-02-19,10:13:12
179 29.880332,121.601705,0,0,40228.4258564815,2010-02-19,10:13:14
180 29.880332,121.601489,0,0,40228.4258796296,2010-02-19,10:13:16
181 29.880333,121.601278,0,0,40228.4259027778,2010-02-19,10:13:18
182 29.880323,121.601068,0,0,40228.4259259259,2010-02-19,10:13:20
183 29.880324,121.60085,0,0,40228.4259490741,2010-02-19,10:13:22
184 29.880329,121.600638,0,0,40228.4259722222,2010-02-19,10:13:24
185 29.88033,121.600404,0,0,40228.4259953704,2010-02-19,10:13:26
186 29.880333,121.600161,0,0,40228.4260185185,2010-02-19,10:13:28
187 29.880334,121.599938,0,0,40228.4260416667,2010-02-19,10:13:30
188 29.880327,121.599751,0,0,40228.4260648148,2010-02-19,10:13:32
189 29.880321,121.599604,0,0,40228.426087963,2010-02-19,10:13:34
190 29.88033,121.599483,0,0,40228.4261111111,2010-02-19,10:13:36
191 29.880337,121.599369,0,0,40228.4261342593,2010-02-19,10:13:38
192 29.88034,121.599231,0,0,40228.4261574074,2010-02-19,10:13:40
193 29.88034,121.599065,0,0,40228.4261805556,2010-02-19,10:13:42
194 29.880343,121.598872,0,0,40228.4262037037,2010-02-19,10:13:44
195 29.88035,121.598678,0,0,40228.4262268519,2010-02-19,10:13:46
196 29.88035,121.598475,0,0,40228.42625,2010-02-19,10:13:48
197 29.880347,121.598279,0,0,40228.4262731481,2010-02-19,10:13:50
198 29.88035,121.598074,0,0,40228.4262962963,2010-02-19,10:13:52
199 29.880343,121.597878,0,0,40228.4263194444,2010-02-19,10:13:54
200 29.880346,121.597685,0,0,40228.4263425926,2010-02-19,10:13:56
201 29.880349,121.597475,0,0,40228.4263657407,2010-02-19,10:13:58
202 29.880347,121.597241,0,0,40228.4263888889,2010-02-19,10:14:00
203 29.880351,121.596999,0,0,40228.426412037,2010-02-19,10:14:02
Normal text file length : 379,604 lines : 6
```

In order to solve the problem described above, we create the module in our project that will be called GPSSimulation which will contain three classes: KafkaMessageProducer, DataSourceRunner, 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

```
95     }
96
97     @ private List<String> validateContent(List<String> rawContent, String outputDir) throws IOException {
98         String file = "invalidLinesLog";
99         final List<String> log = rawContent.stream()
100             .filter(x -> x.contains("777")).collect(Collectors.toList());
101         Files.write(Paths.get(outputDir, file), log, StandardOpenOption.CREATE_NEW);
102         return rawContent.stream().filter(x -> !x.contains("777")).collect(Collectors.toList());
103     }
104 }
```

<< Impl of sol 3.0.4 ps >>

3. Sort all data to maintain chronological order

```
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(), userFolder, "Trajectory"));
    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

```
@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(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(222));
    } catch (InterruptedException ex) {
        Logger.getLogger(SeparateDataSourceSimulator.class.getName()).log(Level.SEVERE, "msg: null, ex);
    }
}
```

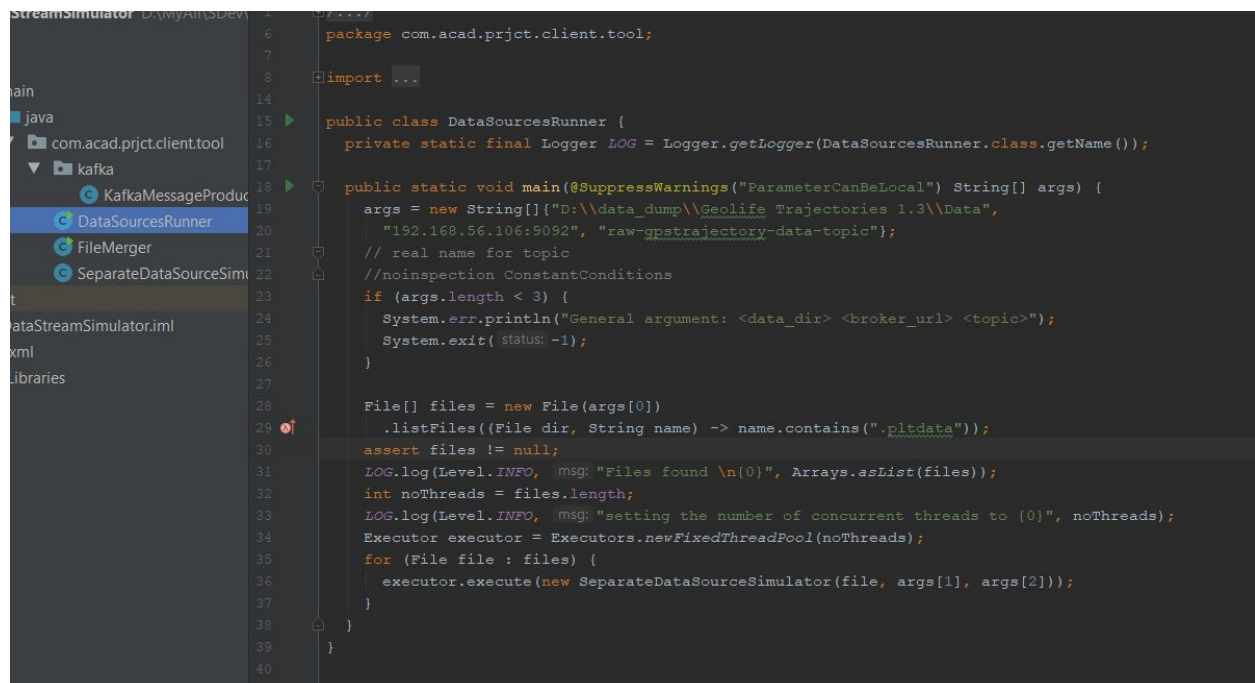
2. Produce all necessary logs to control and monitor data flow
3. Write data (line by line) in the Kafka broker

```
//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
}
}
```


DataSourceRunner



```
1 package com.acad.prjct.client.tool;
2
3 import ...
4
5
6 public class DataSourceRunner {
7     private static final Logger LOG = Logger.getLogger(DataSourceRunner.class.getName());
8
9     public static void main(@SuppressWarnings("ParameterCanBeLocal") String[] args) {
10         args = new String[]{"D:\\data_dump\\Geolife Trajectories 1.3\\Data",
11             "192.168.56.106:9092", "raw-gpstrajectory-data-topic"};
12         // real name for topic
13         //noinspection ConstantConditions
14         if (args.length < 3) {
15             System.err.println("General argument: <data_dir> <broker_url> <topic>");
16             System.exit( status: -1);
17         }
18
19         File[] files = new File(args[0])
20             .listFiles((File dir, String name) -> name.contains(".pltdata"));
21         assert files != null;
22         LOG.log(Level.INFO, msg: "Files found \n{0}", Arrays.asList(files));
23         int noThreads = files.length;
24         LOG.log(Level.INFO, msg: "setting the number of concurrent threads to {0}", noThreads);
25         Executor executor = Executors.newFixedThreadPool(noThreads);
26         for (File file : files) {
27             executor.execute(new SeparateDataSourceSimulator(file, args[1], args[2]));
28         }
29     }
30 }
```

Finally, the DataSourceRunner 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.

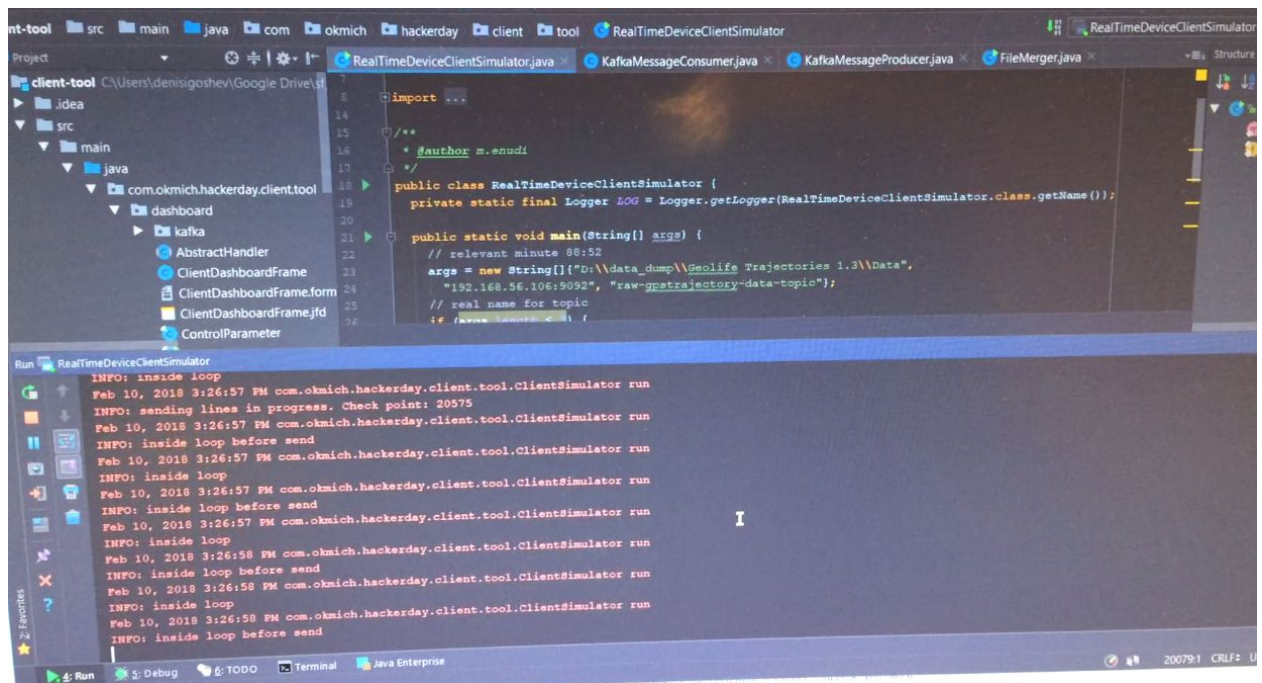


The screenshot shows a web browser displaying the Hortonworks HDP 2.6.0 component version list. The URL is https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.0/bk_release-notes/content/comp_versions.html. The page title is "Official Apache versions for HDP 2.6.0:". The list of components includes:

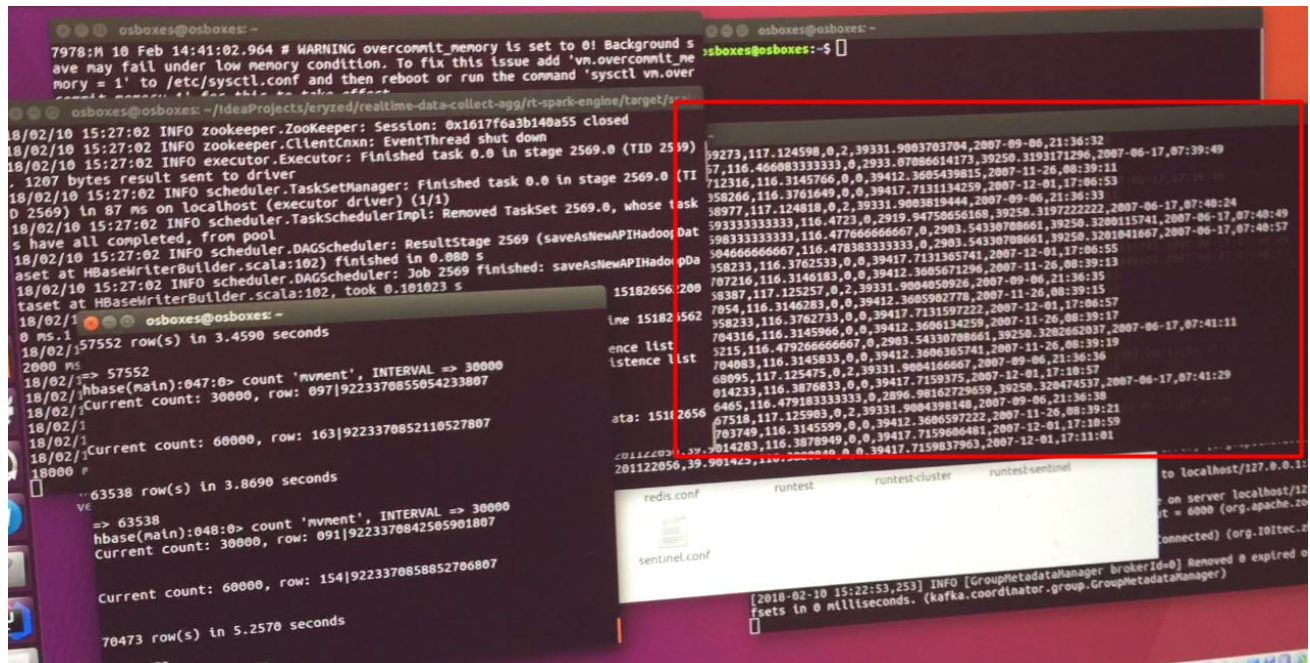
- Apache Accumulo 1.7.0^[1]
- Apache Atlas 0.8.0
- Apache DataFu 1.3.0
- Apache Falcon 0.10.0^[1]
- Apache Flume 1.5.2^[1]
- Apache Hadoop 2.7.3
- • Apache HBase 1.1.2
- Apache Hive 1.2.1
- Apache Hive 2.1.0
- • Apache Kafka 0.10.1.1^[1]
- Apache Knox 0.12.0
- Apache Mahout 0.9.0+^[1]
- Apache Oozie 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



Where the data that is pushing to message broker



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


```

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>>

```

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

def distance(xlat: Float, xlon: Float, ylat: Float, ylon: Float, ell: 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)

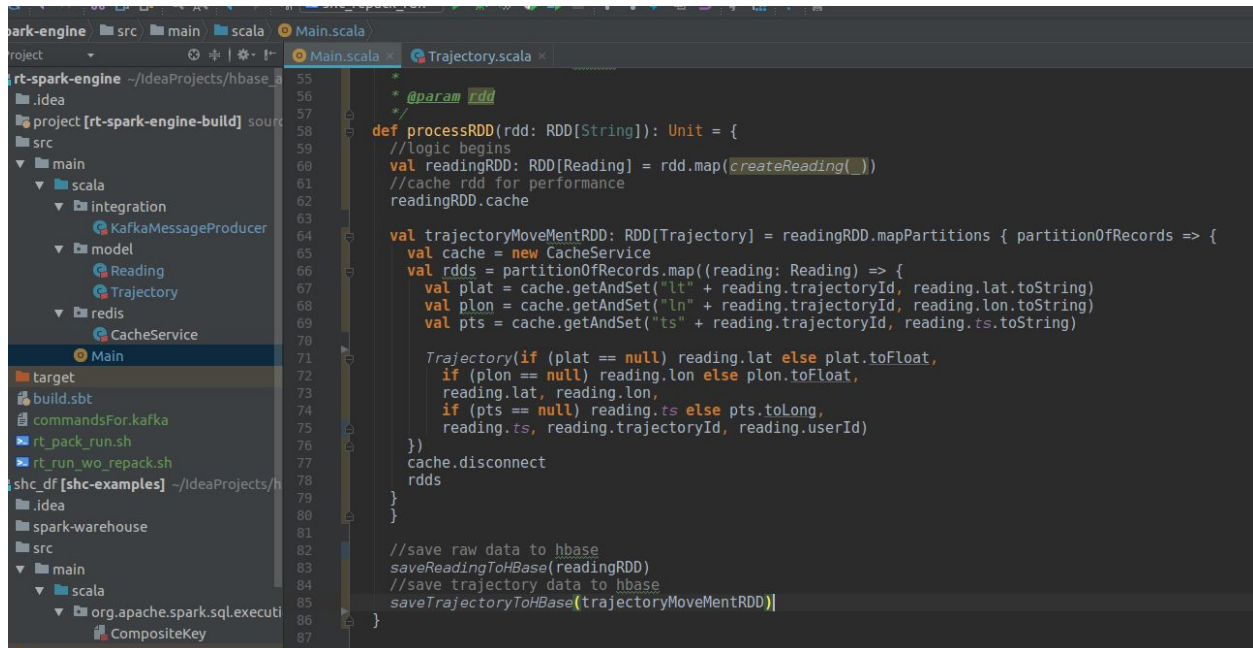
    val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
    var distance = R * c * 1000 // convert to meters
    val height = ell - el2

    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
}

```


The image shows a screenshot of an IDE with a project named 'rt-spark-engine'. The left sidebar shows a file tree with directories like 'main', 'src', 'target', and 'build.sbt'. The main editor window shows the 'Main.scala' file. The code defines a 'processRDD' function that takes an RDD of strings and processes them. It uses a 'CacheService' to store and retrieve data. The code is as follows:

```
55 *  
56 * @param rdd  
57 */  
58 def processRDD(rdd: RDD[String]): Unit = {  
59   //logic begins  
60   val readingRDD: RDD[Reading] = rdd.map(createReading(_))  
61   //cache rdd for performance  
62   readingRDD.cache  
63  
64   val trajectoryMoveMentRDD: RDD[Trajectory] = readingRDD.mapPartitions { partitionOfRecords => {  
65     val cache = new CacheService  
66     val rdds = partitionOfRecords.map((reading: Reading) => {  
67       val plat = cache.getAndSet("lt" + reading.trajectoryId, reading.lat.toString)  
68       val plon = cache.getAndSet("ln" + reading.trajectoryId, reading.lon.toString)  
69       val pts = cache.getAndSet("ts" + reading.trajectoryId, reading.ts.toString)  
70  
71       Trajectory(if (plat == null) reading.lat else plat.toFloat,  
72                 if (plon == null) reading.lon else plon.toFloat,  
73                 reading.lat, reading.lon,  
74                 if (pts == null) reading.ts else pts.toLong,  
75                 reading.ts, reading.trajectoryId, reading.userId)  
76     })  
77     cache.disconnect  
78     rdds  
79   }  
80 }  
81  
82 //save raw data to hbase  
83 saveReadingToHBase(readingRDD)  
84 //save trajectory data to hbase  
85 saveTrajectoryToHBase[trajectoryMoveMentRDD]]  
86 }  
87  
88 }
```

<< 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

```
02/10 15:34:34 INFO scheduler.DAGScheduler: Res
t at HBaseWriterBuilder.scala:102) finished in
02/10 15:34:34 INFO scheduler.DAGScheduler: Job
et at HBaseWriterBuilder.scala:102, took 0.0983
02/10 osboxes@osboxes: ~
ms.1
/02/1
00 ms=> 82706
/02/1 hbase(main):050:0> count 'mvment', INTERVAL => 30000
Current count: 30000, row: 076|9223370860097689807
3/02/1 Current count: 60000, row: 140|9223370858781591807
3/02/1 83055 row(s) in 4.9840 seconds
8/02/1
8/02/1 => 83055
70000 hbase(main):051:0> count 'mvment_by_traj', INTERVAL => 30000
Current count: 30000, row: 20070608232055|9223370855519261807
Current count: 60000, row: 20070901022340|9223370848255752807
VE77825 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
```

id	DISTANCE
27004.25955100550	604.1353791717804
20071211041637	1697.474412312787
20071211055512	12526.205985952374
20071212094726	6135.219517073533
20071215015200	5272.076471602152
124912	2791.16988073774
101100	18808.617766164854
195300	6254.297919670056
022509	1458.1696139364233
103108	340.33457191560547
055038	7823.519804230325
041235	
124344	

```
selected (0.419 seconds)
oentix:localhost>

- user by trajectories
select "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 >>

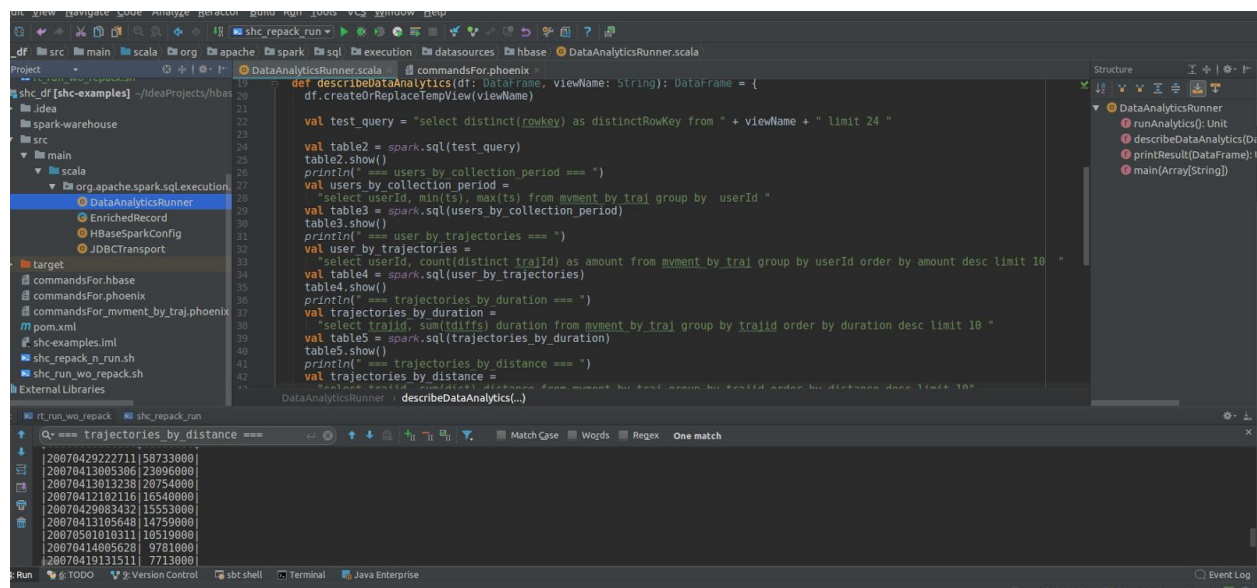
<< Impl of sol 5.0.1 ps >>

<< 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 >>



```
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(duriffs) duration from mvment by traj group by trajId order by duration desc limit 10 "
  val table5 = spark.sql(trajectories_by_duration)
  table5.show()

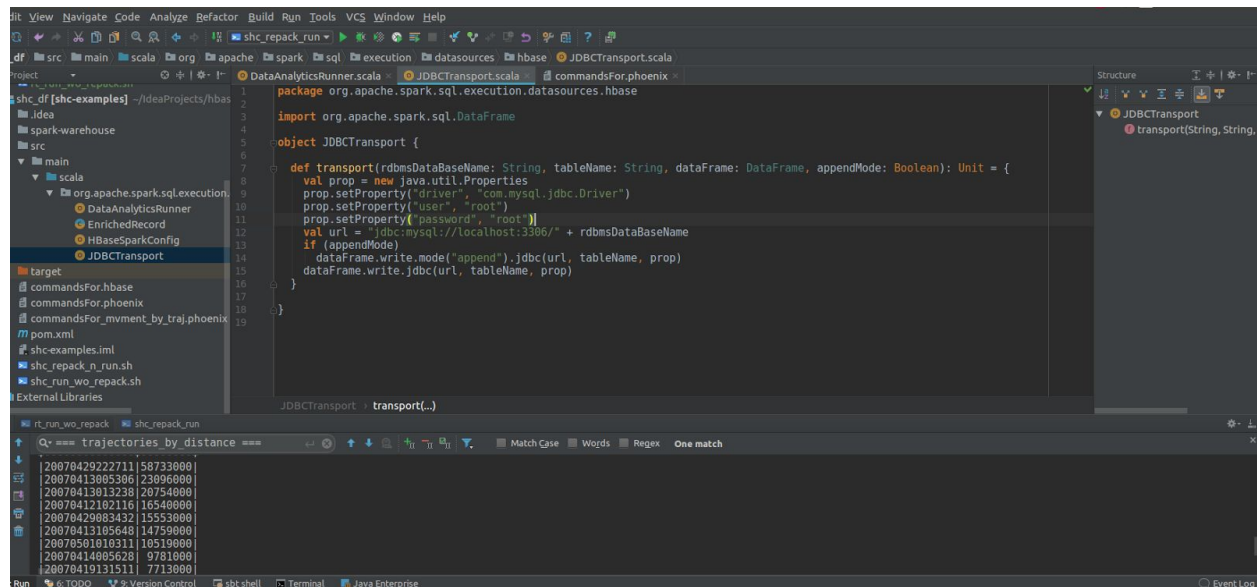
  println(" == trajectories by distance == ")
  val trajectories_by_distance =
    "select trajId, sum(duriffs) duration from mvment by traj group by trajId order by duration desc limit 10 "
  val table6 = spark.sql(trajectories_by_distance)
  table6.show()
}
```

Q: == trajectories by distance ==

```
[20070429222711] 58733000
[20070413005306] 23096000
[20070413013230] 20754000
[20070412102116] 16540000
[20070429083432] 15553000
[20070413105648] 14759000
[20070501010311] 10519000
[20070414005620] 9701000
[20070419131511] 7713000
```

6. Post Analysis

Transfer and save analytical result to RDBMS database



```
1 package org.apache.spark.sql.execution.datasources.hbase
2
3 import org.apache.spark.sql.DataFrame
4
5 object JDBCTransport {
6
7     def transport(rdmsDataBaseName: String, tableName: String, dataframe: DataFrame, appendMode: Boolean): Unit = {
8         val prop = new java.util.Properties
9         prop.setProperty("driver", "com.mysql.jdbc.Driver")
10        prop.setProperty("user", "root")
11        prop.setProperty("password", "root")
12        val url = "jdbc:mysql://localhost:3306/" + rdmsDataBaseName
13        if (appendMode)
14            dataframe.write.mode("append").jdbc(url, tableName, prop)
15        dataframe.write.jdbc(url, tableName, prop)
16    }
17
18 }
19
```

Structure: JDBCTransport > transport(String, String, ...)

trajectories by distance

20070429222711	58733000
20070413005306	23096000
20070413013238	20754000
20070412102116	16540000
20070429083432	15553000
20070413105648	14759000
20070501010311	10519000
20070414085628	9701000
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