

AIM - 3, Homework I

Canevet Gaspard

June 11, 2021

Contents

1	Introduction	3
2	A. Hadoop/MapReduce	3
2.1	Task I	4
2.2	Task II	5
3	B. Spark	7
3.1	Task I	7
3.2	Task II	8
3.3	Task III	9
4	C. Flink	11
4.1	task I	11
4.2	Task II	13
5	D. Flink Streaming	14
5.1	Task I	14
6	Conclusion	16
7	References:	16

1 Introduction

In the coming report, you will find some results regarding the first homework of AIM-3: Scalable Data Science: Systems and Methods course from TU Berlin. In the first part we used hadoop/map reduce technology to extract information from customers and orders dataset. In the second part we used Spark technology to study the customers and orders dataset but also to find the shortest path in a graph. In the third part we used Flink to find the shortest path in a graph with the same input as in second part. Finally, in fourth part we used Flink ability to compute streaming dataset to write a real-time event-based sports analytic application.

2 A. Hadoop/MapReduce

Below, we can find the architecture of the file A. It is made of each output file of task I and II, one pom.xml file for dependencies and program settings, one README that explains how to run codes and a src file with all java codes.

```
C:\Users\gaspa\Desktop\TU_Berlin\AIM-3\HW1_CANEVET\A>tree /F
Folder PATH listing for volume Windows
Volume serial number is 46C8-1B9F
C:..
|
|   output_taskI
|   output_taskII
|   pom.xml
|   README.txt
|
|_ src
   |
   |_ main
      |
      |_ java
         CustomerOrdersDriver.java
         GenericCustomerEntity.java
         JoinMapper.java
         JoinReducerTaskI.java
         JoinReducerTaskII.java
      |
      |_ model
         AbstractCustomerEntity.java
         Customer.java
         CustomerOrdersVO.java
         Orders.java
```

Figure 1: Folder A architecture

The code of this part is based on the gitlab repository [1]. In the model package we can find customers and orders classes. They are the two main classes of the project. It allows to initialise customers and orders object in the mapper. The abstract class AbstractCustomerEntity link customerId feature that is the same in both customers and orders object. Finally, the CustomerOrdersVO is a class made only of customers and orders feature that we want as an output for the task I: customer name, customer address and orders price average.

One good point is that, the mapper is the same for both task. It just reads the line of the file and thanks to number of features it creates a customers or orders object. Nevertheless we have to return the same object type. This is why we created the class GenericCustomerEntity.

Because we implement two different methods for each task we have to create two different Reducer, one for the taskI: JoinReducerTaskI, and one for taskII: JoinReducerTaskII.

Finally we use the same driver class to run a hadoop map reduce task. Our program argument looks like "<input path customer> <input path orders> <output path> <taskI or taskII>".

2.1 Task I

In the first task, we have to implement a mapReduce algorithm to compute the customer's name, address and the average price of orders per customer who has acctbal more than 2000 and for orders placed after 1996-01-01. We can summary this task with an SQL query :

```
SELECT cust.name, cust.adress, AVG(orders.price)
FROM customer AS cust
JOIN orders ON cust.custkey = orders.custkey
WHERE cust.actball > 2000 and orders.orderdate > 1996-01-01
GROUPBY cust.name, cust.address
```

we decided to apply the filter in the reducer. To run the code, we used IntelliJ IDE. Nevertheless we could have compiled the code, exported it as a jar file and computed it with a terminal where Apache hadoop is running. I just find it simpler with IntelliJ. Find below a picture of the configuration.

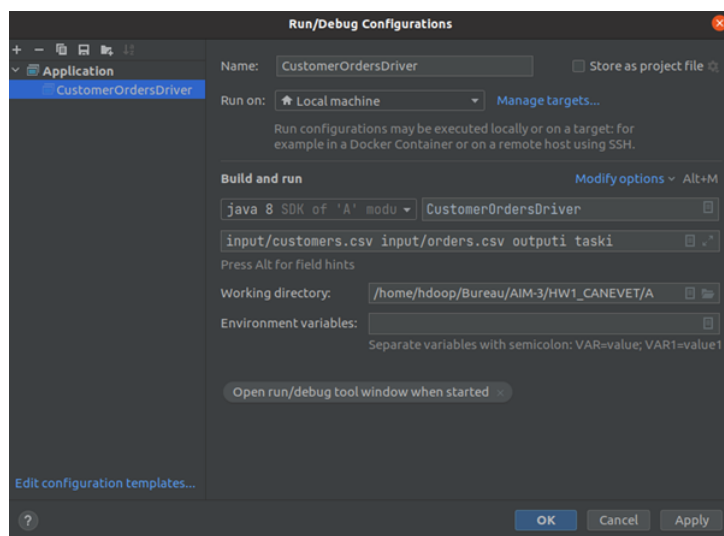


Figure 2: run configuration A task I

Then we share some of the output lines (customer.name, customer.address, AVG(orders.price)). For example the first line name : model.Customer000000004, address : XxVSJsLAGtn, average price : 153786.33187499997

```
model.Customer000000004,XxVSJsLAGtn,153786.33187499997
model.Customer000000007,TcGe5gaZNgVePxU5kRrvXBfksDTea,174347.051
model.Customer000000008,I08108884ymc, 8PrRYBCP1yG38xc8PwmH15,146165.99857142856
model.Customer000000010,4lrEav6KR6PLVcg12ArL Q3rqlzct1 V2,122522.31555555554
model.Customer000000013,nxQk0vJ07PMA59uC3SRSp,119136.56285714285
model.Customer000000014,KXkLmTL23QEA,163822.726
model.Customer000000016,cYlaeMLZMA0Q2 d8W,,58984.8825
model.Customer000000019,vc,3bhlx84W,admlQJvIaXCa2tr,166565.5292857143
model.Customer000000020,3rPk8PqpLj4ne,182094.79333333333
model.Customer000000023,ody W13N78e30C5MpgfmcYss0Wn6TKT,110386.938
model.Customer000000025,Hp8vFQgnHY311HSt8Fe,178176.23166666667
model.Customer000000029,sJ5adtFyAkCK63df2,Vf29zyQWYVE34uh,117421.89750000002
model.Customer000000031,LUACB0bV1aAv6eX0Aebry08 xJvst,169152.28600000002
model.Customer000000032,J02xZz1 UnId,0CtNBLXKJq0Tlp21Q6Zc03J,199692.435
model.Customer000000034,Q669wZ6dnczmT0x509xgE,M2KV,157786.88714285713
model.Customer000000038,a5Ee9568R8RLP Zap7,108075.99571428572
model.Customer000000043,ou5bjHk8lh5fKX3zSso3Z51j9Aa3PoaFd,171642.39454545454
model.Customer000000044,01,d0SPwDu4jo4x,,P85E0dmhZ6vNTBw1,140846.194
model.Customer000000046,eaTXWm10L9,156384.42899999997
model.Customer000000049,cHgAeX7FqrdF7HQW9EwjUa4nxT,68L FKAxz1,158022.7081818182
model.Customer000000050,9S2DYLkzxByyJ1QeTi o,132628.5025
model.Customer000000052,7 Q0q6qq5y9JfV518C71jchJSD0,195075.415
model.Customer000000053,MnaxH2FFfTz8MuCpJyTbZ47Cm4eF00g1b,112435.59142857145
model.Customer000000055,Z1RBR4KNEI HzaIV3a 19ndelrxzDEh8r8pDom,114620.045
model.Customer000000056,BJZYV3Qk4y05B,189525.846
model.Customer000000058,g9ap70k15v9fcXEWjPMyP8Z1RUoh1 T,149714.54499999998
```

Figure 3: Output

Finally we share picture of the terminal in IntelliJ that shows completed work.

```
CustomerOrdersDriver
21/06/06 18:41:58 INFO mapred.Task: Task attempt_local124912221_0001_r_000000_0 is allowed to commit map
21/06/06 18:41:58 INFO output.FileOutputCommitter: Saved output of task 'attempt_local124912221_0001_r_000000_0' to output
21/06/06 18:41:58 INFO mapred.LocalJobRunner: reduce > reduce
21/06/06 18:41:58 INFO mapred.Task: Task 'attempt_local124912221_0001_r_000000_0' done.
21/06/06 18:41:59 INFO mapred.JobClient: map 100% reduce 100%
21/06/06 18:41:59 INFO mapred.JobClient: Job complete: job_local124912221_0001
21/06/06 18:41:59 INFO mapred.JobClient: Counters: 20
21/06/06 18:41:59 INFO mapred.JobClient: Map-Reduce Framework
21/06/06 18:41:59 INFO mapred.JobClient: Spilled Records=33000
21/06/06 18:41:59 INFO mapred.JobClient: Map output materialized bytes=2140750
21/06/06 18:41:59 INFO mapred.JobClient: Reduce input records=16500
21/06/06 18:41:59 INFO mapred.JobClient: Virtual memory (bytes) snapshot=0
21/06/06 18:41:59 INFO mapred.JobClient: Map input records=16500
21/06/06 18:41:59 INFO mapred.JobClient: SPLIT_RAW_BYTES=253
21/06/06 18:41:59 INFO mapred.JobClient: Map output bytes=2101919
21/06/06 18:41:59 INFO mapred.JobClient: Reduce shuffle bytes=0
21/06/06 18:41:59 INFO mapred.JobClient: Physical memory (bytes) snapshot=0
21/06/06 18:41:59 INFO mapred.JobClient: Reduce input groups=1500
21/06/06 18:41:59 INFO mapred.JobClient: Combine output records=0
21/06/06 18:41:59 INFO mapred.JobClient: Reduce output records=718
21/06/06 18:41:59 INFO mapred.JobClient: Map output records=16500
21/06/06 18:41:59 INFO mapred.JobClient: Combine input records=0
21/06/06 18:41:59 INFO mapred.JobClient: CPU time spent (sec)=0
21/06/06 18:41:59 INFO mapred.JobClient: Total committed heap usage (bytes)=713031488
21/06/06 18:41:59 INFO mapred.JobClient: File Input Format Counters
21/06/06 18:41:59 INFO mapred.JobClient: Bytes Read=1892419
21/06/06 18:41:59 INFO mapred.JobClient: FilesystemCounters
21/06/06 18:41:59 INFO mapred.JobClient: FILE_BYTES_READ=6554140
21/06/06 18:41:59 INFO mapred.JobClient: FILE_BYTES_READ=7971250
21/06/06 18:41:59 INFO mapred.JobClient: File Output Format Counters
21/06/06 18:41:59 INFO mapred.JobClient: Bytes Written=47944
```

Figure 4: terminal output A task I

We can see there that the work is done (map 100% map 100%) and that lines are written in the output file (Bytes Written=47944). Finally, the output file named output_taskI is made of 718 lines.

2.2 Task II

In the second task, we have to implement a mapReduce algorithm to compute the name of all customers who did not place any order yet. We can summary this task with an SQL query :

```
SELECT name
FROM customers
WHERE custkey NOT IN (SELECT custkey FROM orders)
```

We also use IntelliJ to run hadoop map reduce algorithm. Find below a picture of the configuration

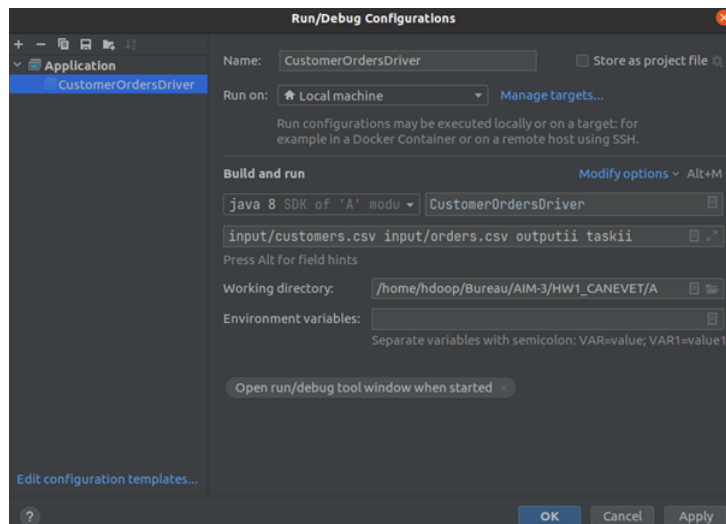


Figure 5: run configuration A task II

Then we share some of the output lines (customers.name). For example the first line name : model.Customer000000003

```
model.Customer#000000003
model.Customer#000000006
model.Customer#000000009
model.Customer#000000012
model.Customer#000000015
model.Customer#000000018
model.Customer#000000021
model.Customer#000000024
model.Customer#000000027
model.Customer#000000030
model.Customer#000000033
model.Customer#000000036
model.Customer#000000039
model.Customer#000000042
model.Customer#000000045
model.Customer#000000048
model.Customer#000000051
model.Customer#000000054
model.Customer#000000057
model.Customer#000000060
```

Figure 6: Output

Finally we share picture of the terminal in IntelliJ that shows completed work.

```
21/06/06 19:18:48 INFO output.FileOutputCommitter: Saved output of task 'attempt_local591207177_0001_r_000000_0' to output11
21/06/06 19:18:48 INFO mapred.LocalJobRunner: reduce > reduce
21/06/06 19:18:48 INFO mapred.Task: Task 'attempt_local591207177_0001_r_000000_0' done.
21/06/06 19:18:48 INFO mapred.JobClient: map 100% reduce 100%
21/06/06 19:18:48 INFO mapred.JobClient: Job complete: job_local591207177_0001
21/06/06 19:18:48 INFO mapred.JobClient: Counters: 20
21/06/06 19:18:48 INFO mapred.JobClient:   Map-Reduce Framework
21/06/06 19:18:48 INFO mapred.JobClient:     Scaled Records=32000
21/06/06 19:18:48 INFO mapred.JobClient:     Map output materialized bytes=2140750
21/06/06 19:18:48 INFO mapred.JobClient:     Reduce input records=16500
21/06/06 19:18:48 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=0
21/06/06 19:18:48 INFO mapred.JobClient:     Map input records=16500
21/06/06 19:18:48 INFO mapred.JobClient:     SPLIT_RAW_BYTES=253
21/06/06 19:18:48 INFO mapred.JobClient:     Map output bytes=2101019
21/06/06 19:18:48 INFO mapred.JobClient:     Reduce shuffle bytes=0
21/06/06 19:18:48 INFO mapred.JobClient:     Physical memory (bytes) snapshot=0
21/06/06 19:18:48 INFO mapred.JobClient:     Reduce input groups=1500
21/06/06 19:18:48 INFO mapred.JobClient:     Combine output records=0
21/06/06 19:18:48 INFO mapred.JobClient:     Reduce output records=500
21/06/06 19:18:48 INFO mapred.JobClient:     Map output records=16500
21/06/06 19:18:48 INFO mapred.JobClient:     Combine input records=0
21/06/06 19:18:48 INFO mapred.JobClient:     CPU time spent (ms)=0
21/06/06 19:18:48 INFO mapred.JobClient:     Total committed heap usage (bytes)=497383040
21/06/06 19:18:48 INFO mapred.JobClient: File Input Format Counters
21/06/06 19:18:48 INFO mapred.JobClient:   Bytes Read=1892419
21/06/06 19:18:48 INFO mapred.JobClient: FilesystemCounters
21/06/06 19:18:48 INFO mapred.JobClient:   FILE_BYTES_WRITTEN=4318816
21/06/06 19:18:48 INFO mapred.JobClient:   FILE_BYTES_READ=7571250
21/06/06 19:18:48 INFO mapred.JobClient: File Output Format Counters
21/06/06 19:18:48 INFO mapred.JobClient:   Bytes Written=12608
```

Figure 7: terminal output A task I

We can see there that the work is done (map 100% map 100%) and that lines are written in the output file (Bytes Written=12608). Finally, the output named output_taskII is made of 500 lines.

3 B. Spark

Below, we can find the architecture of the file B. It is made of each output file of task I and III, one build.sbt file for dependencies and program settings, one README that explains how to run codes and a src file with all scala codes.

```
C:\Users\gaspa\Desktop\TU_Berlin\AIM-3\HW1_CANEVET\B>tree /F
Folder PATH listing for volume Windows
Volume serial number is 46C8-1B9F
C:..
  build.sbt
  output_taskI.csv
  output_taskIII
  README.txt
  src
    main
      scala
        CustomersOrders.scala
        ShortestPath.scala
```

Figure 8: Folder A architecture

We decided to use Scala as the coding language in this part. It is simple to use and to understand. Here, there are two different Scala objects :

- CustomersOrders, the object that compute the task I
- ShortestPath, the object that compute the task III

3.1 Task I

as a reminder, the question of task A(I) : Implement a mapReduce algorithm to compute the customer's name, address and the average price of orders per customer who has acctbal more than 2000 and for orders placed after 1996-01-01. As we said, we can summary this task with an SQL query :

```
SELECT cust.name, cust.adress, AVG(orders.price)
FROM customer AS cust
JOIN orders ON cust.custkey = orders.custkey
WHERE cust.actball > 2000 and orders.orderdate > 1996-01-01
GROUPBY cust.name, cust.address
```

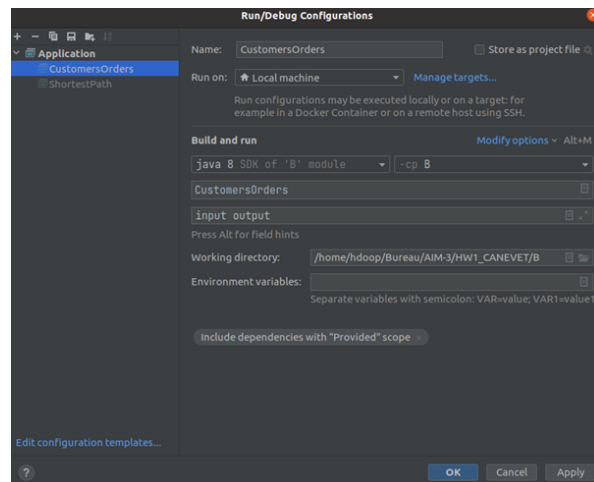


Figure 9: run configuration B task I

Here, we are going to use spark.sql to transform our input into SQL table and use the query. Finally we are going to use IntelliJ to run the code. Our program argument looks like "<input path> <output path>". Find above a picture of the configuration.

Then we share some of the output lines (customers.name, customer.address, AVG(orders.price)). For example the first line name : model.Customer000000004, address : XxVSJsLAGtn, average price : 153786.331875

```
Customer#000000004,XxVSJsLAGtn,153786.331875
Customer#000000007,TcGe5gaZNgVePxU5kRrvXBfkasDTea,174347.051
Customer#000000008,"I0810b80AymmC, 0PrRYBCPlYgJ8xc8PmWhL5",146165.99857142856
Customer#000000010,6LrEaV6KR6PLVcgL2ArL Q3rqzLzcT1 v2,122522.31555555554
Customer#000000013,naXQu0oVj07PM659uC3SRSp,119136.56285714285
Customer#000000014,KXkletMLL2JQEA,163822.726
Customer#000000016,"cYIaeMLZSMA0Q2 d0W,",58984.8025
Customer#000000019,"uc,3bHIX84H,wdrnL0jVsiqXCq2tr",166565.5292857143
Customer#000000020,JrPk8Pqplj4Ne,182094.79333333333
Customer#000000023,0dY W13N7Be30C5MpgfmcYss0Wn6TKT,110306.938
Customer#000000025,Hp86yFQg6HFYS1LH5tBfe,178176.2316666667
Customer#000000029,"sJ5adtffyAkCK63df2,vF25zyQMVE34uh",117421.8975
Customer#000000031,LUACb08viaAv6eX0AebryDB xjVst,169152.286
Customer#000000032,"j02xZzi UmId,DctNBLKXj9q0Tlp21Q6Zc03J",199692.435
```

Figure 10: Output

Finally we share picture of the terminal in IntelliJ that shows completed work.

```
22/06/06 19:07:29 INFO SparkRunner: Finished task 0.0 in stage 0.0 (TID 402), 4309 bytes result sent to driver
22/06/06 19:07:29 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 402) in 4199 ms on 10.0.2.15 (executor driver) (1/1)
22/06/06 19:07:29 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
22/06/06 19:07:29 INFO DAGScheduler: ResultStage 0 (save at CustomersOrders.scala:13) finished in 4.335 s
22/06/06 19:07:29 INFO DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
22/06/06 19:07:29 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
22/06/06 19:07:29 INFO DAGScheduler: Job 2 finished: save at CustomersOrders.scala:13, task 10,178355 s
22/06/06 19:07:29 INFO FileOutputWriter: Write job 7a03a79-c500-403a-a210-7a03a79aeaf committed.
22/06/06 19:07:29 INFO FileOutputWriter: Finished processing chunks for write job 7a03a79-c500-403a-a210-7a03a79aeaf.
22/06/06 19:07:30 INFO SparkContext: Invoking stop() from shutdown hook
22/06/06 19:07:30 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040
22/06/06 19:07:30 INFO MasterTrackerMasterEndpoint: MasterTrackerMasterEndpoint stopped!
22/06/06 19:07:30 INFO MemoryStore: MemoryStore cleared
22/06/06 19:07:30 INFO BlockManager: BlockManager stopped
22/06/06 19:07:30 INFO BlockManagerMaster: BlockManagerMaster stopped
22/06/06 19:07:30 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
22/06/06 19:07:30 INFO SparkContext: Successfully stopped SparkContext
22/06/06 19:07:30 INFO ShutdownHookManager: Shutdown hook called
22/06/06 19:07:30 INFO ShutdownHookManager: Deleting directory /tmp/spark-0773a653-c983-43ed-80c9-f705ed2a0298
Process finished with exit code 0
```

Figure 11: terminal output B task I

To compare outputs from task A(I) and task B(I) we use the number of lines and the start of the document. We can see that both output have 718 lines and look quite similar (We compare the first lines and the last lines). Nevertheless, the average price from task A(I) has more significant digits.

3.2 Task II

Now, we are going to compare both implementations. First in term of expressivity. For the task A(I) we have created eight java classes with more than 50 lines per classes. For the task B(I) we have created one Scala class made of 53 lines. It is clear that Spark implementation simplify code implementation. This is the reason why Spark has been created: To simplify map reduce implementation and we can see there that is verified. Then in term of performance we ran both implementation. the first one took 7s and the second one took one minute. Thus the first one is 7 time faster. You can find below pictures of terminal outputs that expose time computing. We find two main reasons that could explain the differences in time computing. One big reason could be that we use spark.sql and not RDD implementation to compute the task. Then we used Scala for the second task and it is known that java is far more faster. Thus it is crucial to find a trade off between performance and expressivity in this type of study.


```

21/06/06 19:56:30 WARN Utils: Your hostname, canvet-rtbvaldes resolves to a loopback address: 127.0.0.1; using 10.0.2.15 instead (an interface may not
21/06/06 19:56:30 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address)
21/06/06 19:57:30 INFO ShutdownHookManager: Shutdown hook called
21/06/06 19:57:30 INFO ShutdownHookManager: Deleting directory /tmp/spark-07734453-c983-43e0-a0c9-f755ed2dd290
Process finished with exit code 0

```

Figure 12: time computing B task I

3.3 Task III

In the task III we have to write a program that computes the shortest path from node with id 17038 to all other nodes (single source shortest path algorithm). Assuming that each edge from a source node to a destination node costs one, I used the method find on the web page [2] to implement this shortest path algorithm. It is a map reduce based method. first we have to group all source nodes and map data in order to have input like that (sourceNode, weight, state, neighbours, path).

- weight = weight of the path. If edges were differently weighted (positive or negative) it could have been a crucial point. In our case if a point were discovered, it means that we have already found the shortest path to reach that point.
- State = Discover or not
- neighbours = list of neighbours
- path = path to the node (node1 -> node2 -> ...)

Then, while our cluster is not fully explored we find reached nodes (the ones with FINITE weight and UNDISCOVER state). We change their state and we reach their neighbours. At the end we map the output and it looks like that : (initialNode -> node1 ... : path weight)

We are going to use IntelliJ to run the code. Our program argument looks like "<input path> <output path>". We have to add flag -Xss512m to the JVM to perform the algorithm and prevent stackoverflow error. You can find below a picture of the configuration.

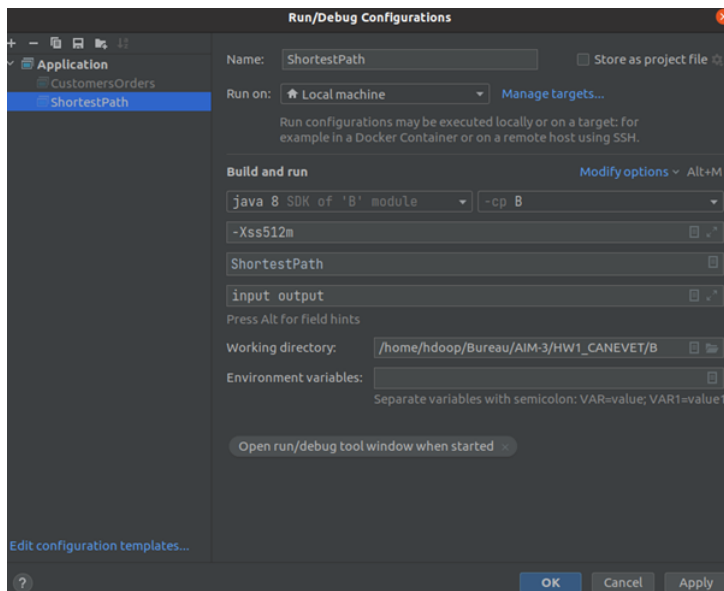


Figure 13: run configuration B task III

Then we share some of the output lines. For example the first line
17038->3466->15931->23344->10956->2350: 5

```
17038->3466->15931->23344->10956->2350: 5
17038->18866->14485->10942: 3
17038->18866->22691->21432->25710->22815->23146: 6
17038->19992->4575->7926: 3
17038->18866->676->12545->5302: 4
17038->17113->25729->11318->11400->11293->20554: 6
17038->17113->2797->20315->1488->7444->14766: 6
17038->21968->7307->14181->15300->19048: 5
17038->18866->676->12545->24057->15395->3212: 6
17038->18866->22691->21432->21491->25569->12166: 6
17038->18866->45->17655->4952->4433: 5
```

Figure 14: Output

Finally we share picture of the terminal in `intelliJ` that shows completed work.

```
at java.lang.Object.wait(Object.java:304)
at java.lang.Thread.join(Thread.java:1252)
at java.lang.Thread.join(Thread.java:1264)
at org.apache.spark.scheduler.AsyncViewQueryExec.doExecute(AsyncViewQueryExec.scala:149)
at org.apache.spark.scheduler.LivelihoodWorker.$anonfun$doStart$1(LivelihoodWorker.scala:228)
at org.apache.spark.scheduler.LivelihoodWorker.$anonfun$doStart$1adapted(LivelihoodWorker.scala:228)
at scala.collection.Iterator.foreach(Iterator.scala:943)
at scala.collection.Iterator.foreach(Iterator.scala:943)
at scala.collection.AbstractIterator.foreach(Iterator.scala:1427)
at scala.collection.IterableLike.foreach(IterableLike.scala:224)
at scala.collection.IterableLike.foreach(IterableLike.scala:224)
at scala.collection.AbstractIterable.foreach(AbstractIterable.scala:56)
at org.apache.spark.scheduler.LivelihoodWorker.$anonfun$doStart$1(LivelihoodWorker.scala:228)
at org.apache.spark.SparkContext.$anonfun$submitText(SparkContext.scala:788)
at org.apache.spark.util.Utils$.tryWithSafeAndFailure(Utils.scala:3357)
at org.apache.spark.SparkContext.submit(SparkContext.scala:1880)
at org.apache.spark.SparkContext.$anonfun$broadcast3(SparkContext.scala:665)
at org.apache.spark.util.SparkShutdownHook.run(ShutdownHookManager.scala:214)
at org.apache.spark.util.SparkShutdownHookManager.$anonfun$runAll(ShutdownHookManager.scala:188)
at scala.runtime.java8.JFunction0$mcVcZ$.apply(JFunction0$mcVcZ$.scala:22)
at org.apache.spark.util.Utils$.logCaughtExceptions(Utils.scala:1676)
at org.apache.spark.util.SparkShutdownHookManager.$anonfun$runAll$1(ShutdownHookManager.scala:188)
at scala.runtime.java8.JFunction0$mcVcZ$.apply(JFunction0$mcVcZ$.scala:22)
at scala.concurrent.Future$.await(Future.scala:213)
at org.apache.spark.util.SparkShutdownHookManager.$anonfun$runAll(ShutdownHookManager.scala:188)
at org.apache.spark.util.SparkShutdownHookManager.$anonfun$runAll(ShutdownHookManager.scala:178) <S Internal Lines>
21/06/26 21:16:16 INFO ShutdownHookManager: Shutdown hook called
21/06/26 21:16:16 INFO ShutdownHookManager: Shutdown hook called
21/06/26 21:16:16 INFO ShutdownHookManager: Deleting directory /tmp/spark-756940c1-a4f2-4376-a76d-3eab082259ef/userfs/bda221a-e340-4831-cede-d4686d2259ef
21/06/26 21:16:16 INFO ShutdownHookManager: Deleting directory /tmp/spark-756940c1-a4f2-4376-a76d-3eab082259ef

Process finished with exit code 0
```

Figure 15: terminal output B task III

We can see there that jobs are done and that blocks are shutdown. To be sure that our output is proper, we can compare the number of line in our output (4158) and clusters information from the website that share the input data. It is written that the largest WCC is made of 4158 nodes. Thus, we could suppose that our output is proper.

4 C. Flink

Below, we can find the architecture of the file C. It is made of output file of task I, one build.sbt file for dependencies and program settings, one README that explains how to run codes and a src file with all scala codes.

```
C:\Users\gaspa\Desktop\TU_Berlin\AIM-3\HW1_CANEVET\C>tree /F
Folder PATH listing for volume Windows
Volume serial number is 46C8-1B9F
C:..
|  build.sbt
|  output_taskI
|  README.txt
|
|_ src
   |_ main
      |_ scala
         ShortestPath.scala
```

Figure 16: Folder C architecture

We decided to use Flink scala api for this part. It is simple to use and it will be easier to compare two algorithms when we use same programming language. For this part we implement only one Scala object:

- ShortestPath, an object that computes the shortest path throughout a graph.

4.1 task I

We are going to use the same method as the task III from part B with the delta iteration trick from Flink api. Thus we need to defines some variables:

- a work set WS
- a solution set S
- a step function f

you can find a small schema from Flink website that explain how the iterate delta trick worked.

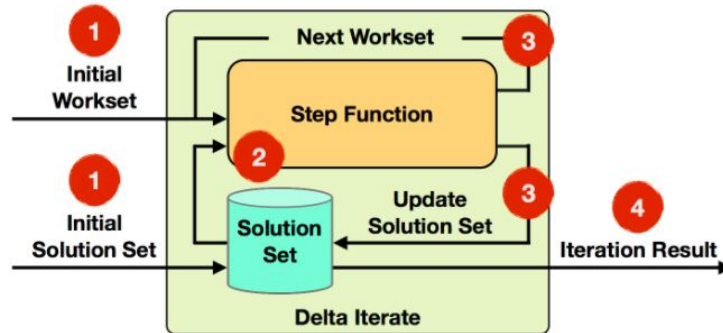


Figure 17: Delta iteration trick [3]

First, we need to define an initial work and solution set to feed the step function. Then the step function will update both set and we will iterate while the workset is not empty or while we do not reach our max iteration number.

We define our workset as the set where we put the newly reached point (the one with a FINITE weight and an UNDISCOVERED state). The initial workset looks like: (17038, 0, UNDISCOVERED, Neighbours,17038)

We define our solution set as the set where we put all the nodes with our special mapping (node,weight,state,neighbours,path). At the end of the Delta iterate we will have to filter our solution set to keep only reached nodes.

Finally our step function need to update:

- The solution set state of reached nodes (UNDISCOVERED -> DISCOVERED)
- The workset with coming reached points (UNDISCOVERED neighbours of newly DISCOVERED nodes)
- Solution set path and weight of coming reached point

Unfortunately, the implementation of deltaIteration function with Scala Flink API do not succeed. This is why I decided to use a basic while loop to perform this task. Nevertheless, I keep the algorithm settings: solution set, workset and step function.

We are going to use IntelliJ to run the code. Our program argument looks like "<inputpath> <output path>". Below, a picture of the configuration.

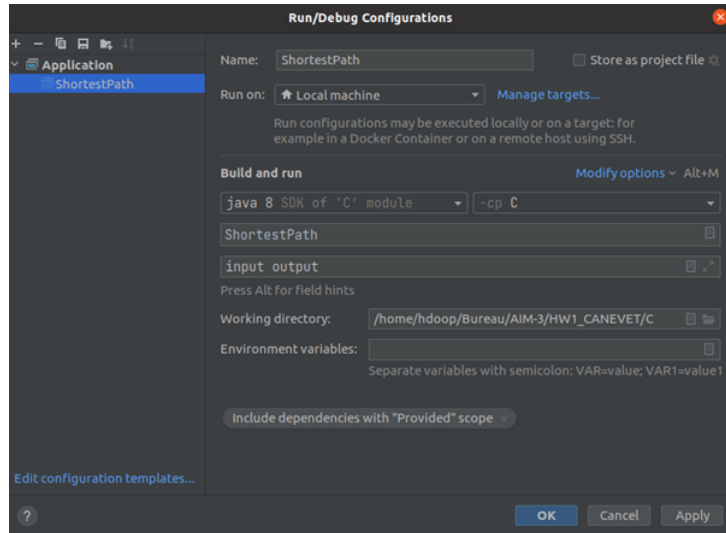


Figure 18: run configuration C task I

Then we share some of the output lines.

```
17038->18866->20391->22555->17306->10004: 5
17038->21968->9017->14746->10022: 4
17038->18866->676->13220->10026: 4
17038->18866->7844->11964->10039: 4
17038->18866->7956->10041: 3
17038->18866->676->17285->21608->10096: 5
17038->18866->676->17285->10113: 4
17038->18866->15003->8612->20100->20236->10116: 6
17038->18866->676->13220->16032->22423->10117: 6
17038->18866->676->12545->5209->1014: 5
17038->18866->7956->21281->950->10153: 5
17038->18866->14485->24924->3651->10162: 5
17038->18866->8503->7350->14265->19059->6494->24781->10183: 8
17038->18866->11785->12101->1023: 4
17038->18866->21912->10243: 3
17038->18866->11785->20595->6823->10247: 5
17038->17113->5862->9248->21125->18215->1025: 6
```

Figure 19: Output

Finally we share picture of the terminal in IntelliJ that shows completed work.

```

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
iteration: 0
iteration: 1
iteration: 2
iteration: 3
iteration: 4
iteration: 5
iteration: 6
iteration: 7
iteration: 8
iteration: 9
iteration: 10
execution time = 28140382904ns
Process finished with exit code 0

```

Figure 20: terminal output C task I

We can see there that jobs are done and the computing time. As we did in the part B(III), we can compare the number of line in our output (4158) and clusters information from the website that share the input data. It is written that the largest WCC is made of 4158 nodes. Thus, we could suppose that our output is proper.

4.2 Task II

Lets study differences between the shortest path implementation perform in task B(III) and C(I). In term of code expressivity, it was far more easier for me to program the Flink part. After finishing the first implementation I was kind of well prepare for the second implementation and thus it was faster and easier. Nevertheless, one great advantage of flink implementation is the algorithm architecture using delta iteration. When I had to design the algorithm it was clearer to use a work set, a solution set, a step function and to loop over it. After all I think that I could have used the same architecture for Spark implementation and It would have saved me lot of time. Finally in term of code performance, it seems that Flink is really faster.

```

21/06/09 19:09:25 INFO SparkHadoopWriter: Job job_202106091909233863437654595986868_4216 committed.
execution time = 87501768352ns21/06/09 19:09:25 INFO SparkContext: Invoking stop() from shutdown hook
21/06/09 19:09:25 INFO SparkUI: Stopped Spark web UI at http://10.0.2.15:4040

```

Figure 21: time computing spark implementation

```

iteration: 9
iteration: 10
execution time = 28140382904ns
Process finished with exit code 0

```

Figure 22: time computing flink implementation

Here we can see that spark implementation take approximately 88s and flink one 28s. Flink implementation is 3 times faster. I really think that my algorithm design is really better in flink. It has surely improved the time computation.

5 D. Flink Streaming

Below, we can find the architecture of the file D. It is made of output file of task I in two parts, one pom.xml file for dependencies and program settings, one README that explains how to run codes and a src file with all java codes.

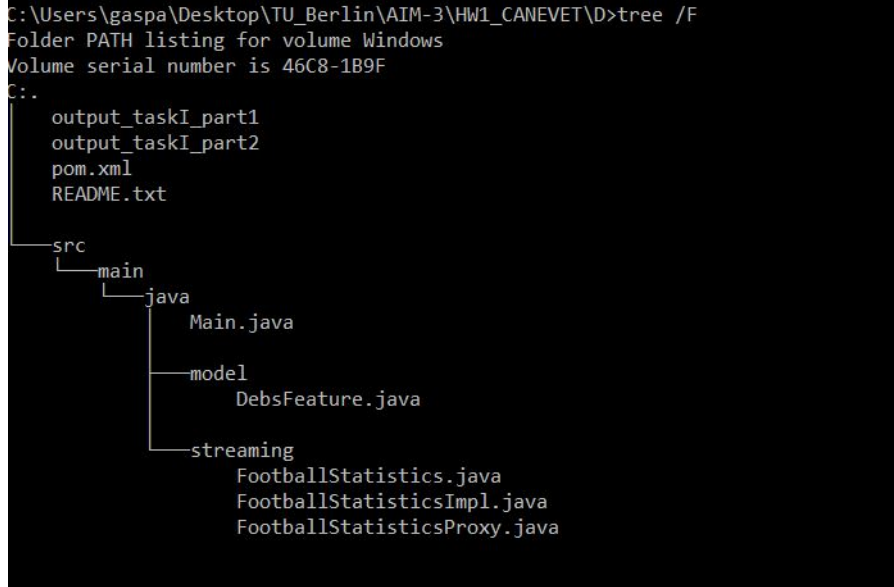


Figure 23: Folder D architecture

For this task, we used a streaming dataset made of sensor data recorded during a football match. you can find deeper information on the paper release website [5]. The total produced data rate reaches about 15,000 position events per second. The data has the following schema:

$$\text{sid, ts, x, y, z, } |v|, |a|, vx, vy, vz, ax, ay, az$$

Where sid is a sensor id, ts is a timestamp in picoseconds, x, y, z describe the position of the sensor in mm (the coordinate (0,0,0) corresponds to the center of the football field), $|v|$ (in $\mu\text{m/s}$) describes the absolute velocity and vx, vy, vz are the directed velocity vectors with a normalized size of 10,000. Similarly, $|a|$ (in $\mu\text{m/s}^2$), ax, ay, az describe the absolute acceleration and the directed acceleration vectors with a normalized size of 10,000.

The source code of this part is based on the gitlab repository [4]. the model DebsFeature is a Pojo in order to load streaming events. Then we will have to put our code for task I and II respectively in functions highestAverage and writeAvertedGoalEvents located in FootballStatisticsImpl class. Finally, we will have to run the main function in the class Main in order to obtain our results. Nevertheless I only have the time to perform the first task.

5.1 Task I

In the first task we have to compute the average distance that Player A1 runs in a five-minute window, where the duration between consecutive windows is one minute. We also have to report the highest average distance among all windows for Player A1. Some hints were given for this task.

First we have to filter incoming events in order to have only sensor from A1 player during the match. Thus we have to keep sensors 16 and 47, timestamps between 10 753 295 594 424 116 and 12 557 295 594 424 116 for the first half and timestamps between 13 086 639 146 403 495 and 14 879 639 146 403 495 for the second half.

Then we used sliding window method with five minutes duration each 1 minutes to compute a

moving average. The output of my moving average is a tuple (startWindowTimestamp, EndWindowTimestamp, WindowDistanceAverage, emptyStringKey). The last element of the tuple will allow us to key windows and display only the maximum average throughout all windows. Finally to compute distance we used euclidian distance with input x and y like below:

$$d = \sqrt{x^2 + y^2} \quad (1)$$

We are going to use IntelliJ to run the code. Our program argument looks like "<inputpath>". Below, a picture of the configuration.

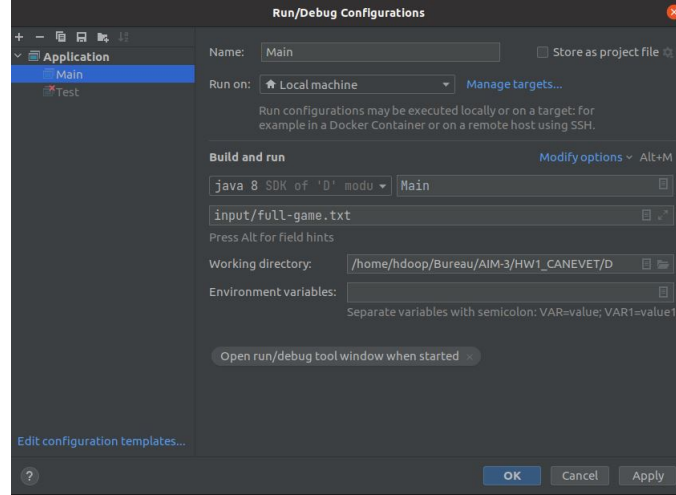


Figure 24: run configuration D task I

Then we share some of the output lines. (statWindowTime,endWindowTime,HighestAVG)

```
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
10753297603582109,10782300748491212,38670.87871913445,
```

Figure 25: Output

Finally we share picture of the terminal in IntelliJ that shows completed work. there is no crucial information, just that process is successfully finished

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Process finished with exit code 0
```

Figure 26: terminal output D task I

6 Conclusion

To conclude, I did not manage to do everything. I did not have the time and still the motivation to do everything. It was a really tough assignments but I learned a lot of new things. Maybe some basic example could have been shared before the homework in order to better understand the purpose of each technology. For instance some simple example on Spark and Flink could have been really useful.

7 References:

- [1]https://gitlab.tubit.tu-berlin.de/cschulze/AIM3-SQL_in_MapReduce/tree/master/src/main/java
- [2]<https://adhoop.wordpress.com/2012/03/06/shortest-path-algorithm-in-mapreduce/>
- [3]<https://ci.apache.org/projects/flink/flink-docs-release-1.13/docs/dev/dataset/iterations/>
- [4]<https://gitlab.tubit.tu-berlin.de/AIM3-SDS/ComputationalExercises>
- [5]<https://dl.acm.org/doi/10.1145/2488222.2488283>