Architectures for big data – Exam 20/01/2022

Dataset

- **Transactions**
    - **Description**
      set of transactions made on the stock market by any user on any given company. Each user can "Buy" or "Sell" (*Action* values) a given *Volume* of stock of a given CompanySymbol.
    - **Schema**
      TransactionID:Int | Timestamp:Datetime | UserID:String | CompanySymbol:String | Volume:Int | Action:String

- **Prices**
    - **Description**
      Everytime a transaction is completed, the price (i.e., *ValuePerUnit_EURO)* of stocks of a given CompanySymbol could change. Everytime it changes, the new value is stored in the **Prices** table
    - **Schema**
      CompanySymbol:String | Timestamp:Datetime | ValuePerUnit_EURO: Float

- **CompanyInfo**
    - Description
      Additional Information on any company
    - Schema
      CompanySymbol:String | CompanyName:String | Address: String

Questions

Q1 – 6 points

Define the type of each table (Log or Registry) and architect a strategy to integrate them through a CDC job. You can propose a pseudo-code solution for the CDC job, a SQL based solution, a mix of the two.

LOG – LOG - REGISTRY

I need a batch process that read the content of the table and get new lines as diff with the previous reading.

First of all I need a file/structure to save the previous state: the state could be the maximum timestamp for log-like tables or the list of Keys and Values for a registry like table.

LOG CDC

I need to get this information in a SQL statement where I can inject the timestamp value.

Transaction is a log table, so the query could be

```
Select * from Transaction where Timestamp > $MAX_TS ORDER BY Timestamp
```

Same story for Prices

```
Select * from Prices where Timestamp > $MAX_TS ORDER BY Timestamp
```

REGISTRY CDC

For Registry tables I need to make a full table scan, fetch each line one by one comparing with the KHASH-HASH list generated in the previous iteration.

CompanyInfo is a registry, so I need first to save on a sync file the content. To reduce the space I save only one hash for the values of keys and one hash for the value of other columns.

Then I apply same hashing strategy to the actual extraction.

Line by line I check the status

```
getHash(line,keyColumns):-> return a md5 hash considering only non-key columns
getKhash(line,keyColumns):-> return a md5 hash considering only key columns

previousHashKhashMap = loadPreviousRun()
#{'khash1':'hash1','khash2':'hash2',…}

for line in allLines:
        hash = getHash(line,keyColumns)
        kHash = getKhash(line,keyColumns)
        if kHash not in previousHashKhashMap ➔ insert
        elif hash != previousHashKhashMap.get(kHash) ➔ update
```

This solution ignore deletes.

Q2 – 3 points each

Assumption
you have already read the 3 tables and you have a session with PricesRDD, TransactionsRDD, and CompanyInfoRDD where you find any cdc created over time.

The aesthetic part of the code (e.g., names used for variables) as well as the distributability will be considered.

Write the code to compute:

1. Total Number of transactions
   TransactionsRDD.count()
2. Number of Transactions done by the user "HAL9000"
   TransactionsRDD.filter(lambda x: x.get("UserID") == "HAL9000").count()
3. Number of transactions per day
   transactionsPeeDay = TransactionsRDD.map(lambda x: (x.get("Timestamp").isocalendar(),1)).reduceByKey(lambda x,y: x+y).persist()
   transactionsPeeDay.count()
4. Average Daily Transactions per company (i..e, On average, how much transaction each company does every day) during the week 42 of 2021
   week42 = TransactionsRDD.filter(lambda x: x.get("Timestamp").isocalendar()[1] == 42).map(lambda (x: (x.get("CompanySymbol"),1/5)).reduceByKey(lambda x,y: x+y).persist()
   week42.count()

   Total Amount of Euro spent by each user

```
transactionWithPricesRdd = transactionRdd.map(lambda x: (x.get("companySymbol"),x))\
                              .join(PricesRdd.map(lambda x: (x.get("companySymbol"),x)))\
                              .map(lambda x: {"companySymbol":x[0],"userId":x[1][0].get("userId"),
                                   "ID":x[1][0].get("transactionId"), "Action":x[1][0].get("Action"),
                                   "volumes":x[1][0].get("volumes"),
"pricePerUnit":x[1][1].get("pricePerUnit"),

     "moneyValue":x[1][0].get("volumes")*x[1][1].get("pricePerUnit")
                                   "ts":x[1][0].get("timestamp"),
                                   "deltaTs":x[1][0].get("timestamp")-x[1][1].get("timestamp")
                                   })

transactionWithPriceRdd = transactionWithPricesRdd.filter(lambda x: x.get("deltaTs")>0)\
                                   .map(lambda x: (x.get("transactionId"),(x)))\
                                   .reduceByKey(lambda x: x if x.get("deltaTS")<y.get("deltaTS")
else y)
                                   .map(lambda x: x[1]).persist()

spentByUserRdd = transactionWithPriceRdd.filter(lambda x: x.get("Action")=="Buy")\
     .map(lambda x: x.get("UserId"),x.get("moneyValue ")).reduceByKey(x+y).persist()

spentByUserRdd.count()
```

5. Value of the portfolio (the value of stocks currently hold) of each user

```
amountOfStocksRdd = transactionRdd.map(lambda x: ((x.get("userId"),x.get("companySymbol"),
                                 x.get("Volume") if x.get("Action") == "Buy" else -x.get("Volume")))\
                                 .reduceByKey(lambda x,y: x+y).map(lambda x: x[0][1],(x[0][0],x[1])).persist()

lastValueRdd = PriceRdd.map(lambda x: (x.get("companySymbol"),x))\
                                           .reduceByKey(lambda x,y: x if x.get("Timestamp")>y.get("Timestamp")
else y)\
                                           .persist()

currentPortfolioRdd = amountOfStocksRdd.join(lastValueRdd)\
                                                               .map(lambda x: {"UserId":x[1][0][0],
"Value":x[1][0][1]*x[1][1].get("Price")})\
                                                               .map(lambda x: x.get("UserId",x.get("Value")))\
                                                                                           .reduceByKey(lambda x,y:
x+y).persist()

currentPortfolioRdd.count()
```