DSCI551 Final (Tuesday) - Results

Attempt 1 of 1

Written May 9, 2023 2:01 PM - May 9, 2023 4:01 PM

Your quiz has been submitted successfully.

Attempt Score **95.5 / 100 - A**
Overall Grade (Highest Attempt) **95.5 / 100 - A**

0. Note

All your codes answered in this exam are expected to run properly.

1. Hadoop MapReduce, 30 points

Consider the following tables. Note the primary key of each table is underlined.

Purchase(buyer, seller, product, store, price)
Product(name, maker, category)
Person(name, city, phone)

Sample data:
Purchase("John", "David", "Iphone12", "store1", 500)
Purchase("David", "Mary", "Iphone11", "store2", 300)
Purchase("John", "Mary", "T14s", "store2", 1000)
Product("Iphone12", "Apple", "Cell phone")
Product("Iphone11", "Apple", "Cell phone")
Product("T14s", "Lenovo", "Laptop")
Person("John", "LA", "626-123-4567")
Person("David", "SFO", "909-456-1234")
Person("Mary", "LA", "303-312-8867")

Note that the above data is just sample data and your codes in all questions in this exam on these tables should work on suitable data of the tables, not just the sample data.

Now suppose data of these tables are stored in CSV files: purchase.csv, product.csv, and person.csv. For example, there are rows in sample purchase.csv (note strings are not quoted):
John,David,Iphone12,store1,500
David,Mary,Iphone11,store2,300
John,Mary,T14s,store2,1000

Consider writing a Hadoop MapReduce program to answer the following SQL query using the provided CSV files:

```sql
Select buyer, avg(price)
From Purchase
Where store = 'store1' or store = 'store2'
Group by buyer
Having count(*) > 1
```

Question 1 — 7 / 8 points

Write pseudocode for the map function. Be sure to indicate input arguments, steps, and include the output in the function.

```
map(line):
  fields = line.split(',')
  if fields[3] == 'store1' or fields[3] == 'store2':
    key = fields[0]  # buyer is the key
    value = float(fields[4])
```

output(key, value)

Input: line from purchase.csv which is a purchase record
Output: key = buyer, value = price

Steps:
1. Split the input value by comma and assign values to variables: buyer, seller, product, store, price
2. Check if store is equal to 'store1' or 'store2'
3. If yes, emit key-value pair (buyer, price)
4. Else, do not emit anything.

▼ Hide question 1 feedback

input is offset and line -1

Question 2 **5 / 5 points**

What are the input and output key-value pairs to the map function for the sample data? Assume that the input key is the row offset instead of byte offset, so the offset for the first row is zero, the offset for the second row is 1, and so on.

input:
Key: 0
Value: "John,David,Iphone12,store1,500"

output:
Key: "John"
Value: 500.0

input:
Key: 1
Value: "David,Mary,Iphone11,store2,300"

output:
Key: "David"
Value: 300.0

input:
Key: 2
Value: "John,Mary,T14s,store2,1000"

output:
Key: "John"
Value: 1000.0

Question 3 **8 / 8 points**

Assume no combiner is used. Write pseudocode for the reduce function. Be sure to indicate input arguments, steps, and include the output in the function.

input: key-value pair where the key - buyer and value - list of prices.
output: key-value pair where the key - buyer and value - avg price

Pseudocode:

```
reduce(key, values):
  sum = 0.0
  count = 0
  for value in values:
    sum += value
    count += 1
  if count > 1:
    average = sum / count
    output(key, average)
```

Steps:
1. Initialize sum_price and count to zero.

2. Loop through each tuple in the list of values:
Add the price to sum_price.
Add the count to count.
3. If the count is greater than 1, calculate the average price by dividing the sum_price by the count.
4. If the average price is not zero, emit a key-value pair where the key is the buyer and the value is the average price.
Output: (buyer, avg_price)

Question 4 **4 / 4 points**

What are the input and output key-value pairs to the above reduce function for the sample data?

input:
Key: "John"
Value: [500.0, 1000.0]

Key: "David"
Value: [300.0]

output:
Key: "John"
Value: 750.0

Question 5 **2 / 5 points**

Now assume that a combiner is used. Describe the job of combiner and its benefits. Describe the changes to the reduce function.

- Combiner combines the intermediate key-value pairs produced by the map function before sending them to the reducer
- Receives a subset of the key-value pairs emitted by the map function
- Reduces the amount of data that needs to be transferred to the reducer

Benefits:
1. Reducing network traffic: Reduces the amount of data transferred over the network from the mappers to the reducers, which can significantly speed up the job execution time.
2. Reducing disk I/O: By reducing the amount of data sent over the network, combiners can also reduce the amount of disk I/O needed for reading the intermediate data on the reducers.
3. Improved performance: By aggregating intermediate key-value pairs on the map nodes, combiners can improve the overall performance of the MapReduce job by reducing the amount of processing needed by the reducers.

Changes:
Reduce function still receives the same input key-value pairs, but they are now partially aggregated.

▼ Hide question 5 feedback

wrong changes in the reduce function -3

please brush on the course materials for a better understanding of the combiner

2. Query execution, 20 points

Consider the following query:

```SQL
Select *
From Purchase join Product
where Purchase.product = Product.name
```

Suppose the Purchase table occupies 1000 blocks, the Product table 2000 blocks. Assume that there are 101 pages available for the above join.

Question 6 **10 / 10 points**

Describe a **sort-merge** join algorithm for the above join. Assume that 100 pages are used for the sorting phase of the algorithm. Be sure to indicate the steps, and input and output of each step (e.g., how many runs and the size of runs, etc.).

We use 100 blocks for loading at once.
Purchase table = 1000 blocks
Product table = 2000 blocks
M = 101

Sort-merge is a 2 pass algorithm. First we sort both the tables.

Pass 1 (Sorting):
10 runs with 100 blocks/run to sort Purchase
20 runs with 100 blocks/run to sort Product

Cost for read and write {Purchase, Product} = 2B(Purchase) + 2B(Product)

Since total number of runs < M-1 i.e 30<100, no additional step will be required.

Pass 2 (Merging):
Cost for merging {Purchase, Product} = B(Purchase) + B(Product)

Question 7 **2 / 2 points**

What is the cost of the join algorithm on the above query?

Total Cost
= 3B(Purchase) + 3B(Product)
= 3*1000 + 3*2000
= 9000 block I/O's

2.1 [8 points]

Now suppose the Purchase table has 5,000 blocks, and the Product table has 20,000 blocks.

Question 8 **6 / 6 points**

Describe the change to the algorithm and the steps. Again, assume that 100 pages are used for the sorting phase of the algorithm

We use 100 blocks for loading at once.
Purchase table = 5000 blocks
Product table = 20000 blocks
M = 101

Sort-merge is a 2 pass algorithm. First we sort both the tables.

Pass 1 (Sorting):
50 runs with 100 blocks/run to sort Purchase
200 runs with 100 blocks/run to sort Product

Cost for read and write {Purchase, Product} = 2B(Purchase) + 2B(Product)

Since total number of runs > M-1 i.e 250>100, additional step will be required for Product where we are merging 200 runs => 200/100 => 2 runs.
Additional Cost for Read and write Product = 2B(Product)

Pass 2 (Merging):
Cost for merging {Purchase, Product} = B(Purchase) + B(Product)

Question 9 **2 / 2 points**

What is the cost of the algorithm on the new tables?

Total Cost
= 3B(Purchase) + 5B(Product)
= 3*5000 + 5*20000
= 115000 blocks I/O's

3. SQL & Spark DataFrame, 25 points

Now suppose the CSV files in Question 1 have headers. For example, purchase.csv with header:

buyer,seller,product,store,price
John,David,Iphone12,store1,500
David,Mary,Iphone11,store2,300
John,Mary,T14s,store2,1000

Suppose that the data are loaded into Spark for processing as follows.

```python
import pyspark.sql.functions as fc
purchase = spark.read.csv('purchase.csv', header=True, inferSchema=True)
product = spark.read.csv('product.csv', header=True, inferSchema=True)
person = spark.read.csv('person.csv', header=True, inferSchema=True)
```

For each of the following questions, write an SQL query and a Spark DataFrame script to answer the question. You can assume that SQL query can include set operations.

Question 10 — 6 / 6 points

Find out who bought the largest number of products. Assume that there is only one such person.

i. [3 points] SQL

```
SELECT buyer, COUNT(DISTINCT product) AS num_prods
FROM purchase
GROUP BY buyer
ORDER BY num_prods DESC
LIMIT 1;
```

ii. [3 points] DataFrame

```
from pyspark.sql.functions import countDistinct, desc

purchase.groupBy("buyer").agg(countDistinct("product").alias("num_prods")).orderBy(
  desc("num_prods")
).limit(1).show()
```

Question 11 — 6 / 6 points

Find out who bought products in store 1 but not store 2.

i. [3 points] SQL

```
SELECT buyer
FROM purchase
WHERE store = 'store1' AND buyer NOT IN (
  SELECT buyer
  FROM purchase
  WHERE store = 'store2'
)
```

ii. [3 points] DataFrame

```
from pyspark.sql.functions import col

purchase1 = purchase.filter(col('store') == 'store1')
purchase2 = purchase.filter(col('store') == 'store2')

buyers = purchase1.select('buyer').subtract(purchase2.select('buyer'))
buyers.show()
```

Question 12 — 7 / 7 points

Find out the average price for each category of products **sold in store 1**.

i. [3 points] SQL

```
SELECT category, AVG(price) AS avg_price
FROM purchase p
JOIN product pr ON p.product = pr.name
WHERE store = 'store1'
GROUP BY category;
```

ii. [4 points] DataFrame

```
from pyspark.sql.functions import avg, col

purchase1 = purchase.filter(col("store") == "store1")
avg_price = (
  purchase1.join(product, purchase1.product == product.name)
  .groupBy("category")
  .agg(avg("price").alias("avg_price"))
)

avg_price.show()
```

Question 13 **6 / 6 points**

Find out how many products were sold at store 1 to someone living in LA.

i. [3 points] SQL

```
SELECT COUNT(*) AS num_sold
FROM purchase p
JOIN person pe ON p.buyer = pe.name
WHERE store = 'store1' AND city = 'LA';
```

ii. [3 points] DataFrame

```
from pyspark.sql.functions import count, col

num_sold = (
  purchase.join(person, purchase.buyer == person.name)
  .filter((col("store") == "store1") & (col("city") == "LA"))
  .agg(count("*").alias("num_sold"))
)
num_sold.show()
```

4. Spark RDD, 25 points

Now suppose we use RDD of dataframes created in the previous question as follows.

Python

```
purchaseRDD = purchase.rdd
productRDD = product.rdd
personRDD = person.rdd
```

For each of the following questions, write an SQL query and a Spark RDD script to answer the question.

Question 14 **5 / 5 points**

Find out names of people whose phone number contains "123" as a substring. Return names only.

i. [2 points] SQL

```
SELECT name
FROM person
WHERE phone LIKE '%123%';
```

ii. [3 points] RDD

```
person_names = personRDD.filter(lambda row: "123" in row.phone).map(
  lambda row: row.name
)
person_names.collect()
```

Question 15 **4.5 / 5 points**

Find out who bought the smallest number of products. Assume that there is only one such person. Output only the buyer's name.

i. [2 points] SQL

```
SELECT buyer
FROM Purchase
GROUP BY buyer
HAVING COUNT(*) = (
  SELECT MIN(prod_count)
  FROM (
    SELECT COUNT(*) AS prod_count
```

```
    FROM Purchase
    GROUP BY buyer
  ) p
);
```

ii. [3 points] RDD

```
purchase_count = purchaseRDD.map(lambda row: (row.buyer, 1))
purchase_count = purchase_count.reduceByKey(lambda a, b: a + b)
min_count = purchase_count.map(lambda x: x[1]).min()
min_buyers = purchase_count.filter(lambda x: x[1] == min_count)
min_buyers_names = min_buyers.map(lambda x: x[0])
```

▼ Hide question 15 feedback

need collect

Question 16 **5 / 5 points**

Find out who bought products **in both store 1 and store 2**. Output the same buyer (name) only once.

i. [2 points] SQL

```
SELECT DISTINCT buyer
FROM purchase
WHERE store = 'store1'
AND buyer IN (
  SELECT buyer
  FROM purchase
  WHERE store = 'store2'
)
```

ii. [3 points] RDD

```
store1_purchases = purchaseRDD.filter(lambda row: row.store == "store1")
store2_purchases = purchaseRDD.filter(lambda row: row.store == "store2")

buyers = set(store1_purchases.map(lambda row: row.buyer).distinct().collect()) & set(
  store2_purchases.map(lambda row: row.buyer).distinct().collect()
)
for buyer in buyers:
  print(buyer)
```

Question 17 **5 / 5 points**

Find out the **average** price for each category of products sold in store 1. Output the category and its average price.

i. [2 points] SQL

```
SELECT product.category, AVG(purchase.price) AS avg_price
FROM purchase
JOIN product ON purchase.product = product.name
WHERE purchase.store = 'store1'
GROUP BY product.category
```

ii. [3 points] RDD

```
product_RDD = productRDD.map(lambda row: (row["name"], row))
purchase_RDD = purchaseRDD.map(lambda row: (row["product"], row))
joined_RDD = purchase_RDD.join(product_RDD)

joined_RDD.filter(lambda row: row[1][0].store == 'store1')
.map(lambda row: (row[1][1].category, (row[1][0].price, 1)))
.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
.mapValues(lambda x: x[0] / x[1]).collect()
```

Question 18 **5 / 5 points**

Find out how many products were sold at store 1 or store 2 to someone living in LA. Return the number of

products sold.

i. [2 points] SQL

```
SELECT COUNT(*) as product_count
FROM purchase
JOIN person ON purchase.buyer = person.name
WHERE (purchase.store = 'store1' OR purchase.store = 'store2')
AND person.city = 'LA'
```

ii. [3 points] RDD

```
person_RDD = personRDD.map(lambda row: (row["name"], row))
purchase_RDD = purchaseRDD.map(lambda row: (row["buyer"], row))
joined_RDD = purchase_RDD.join(person_RDD)

joined_RDD.filter(
  lambda row: (row[1][0].store == 'store1' or row[1][0].store == 'store2')
  and row[1][1].city == 'LA'
).count()
```

Done