DSCI551 Final (Tuesday) - Results

Attempt 1 of 1

Written May 9, 2023 2:01 PM - May 9, 2023 4:18 PM

Your quiz has been submitted successfully.

Attempt Score	**96.5 / 100 - A**
Overall Grade (Highest Attempt)	**96.5 / 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 8 / 8 points

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

```
def map(key, value):
 toks = split(value, ",")

 // Check if the store is either store1 or store2
 if (toks[3] == "store1" OR toks[3] == "store2"):

  // Get the buyer and price fields
  buyer = toks[0]
  price = toks[4]

  // Output the key-value pair (buyer, price)
  output(buyer, price)
```

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 -> Output)

(0, "John,David,Iphone12,store1,500") -> ("John", 500.0)
(1, "David,Mary,Iphone11,store2,300") -> ("David", 300.0)
(2, "John,Mary,T14s,store2,1000") -> ("John", 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.

```
reduce(key, values):

 // Count the number of purchases for the buyer
 count = 0
 for price in values:
  count = count + 1

 // Compute the average price if count > 1
 if count > 1:
  sum = 0
  for price in values:
    sum = sum + price
  avg_price = sum / count

  // Output the key-value pair (buyer, avg_price)
  output(key, 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-value pairs
"David" -> [300]
"John" -> [500, 1000]

// Output key-value pair
"John" -> 750.0
```

Question 5 3 / 5 points

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

A combiner takes the output of the map function as input and performs a local aggregation before sending the data to the reduce function. The job of the combiner is to reduce the amount of data transferred from the map phase to the reduce phase by performing local aggregation on the output key-value pairs of the map function. Its benefits are:
1. Reduced amount of data transferred over the network from mapper to reducer, which improves the overall performance of the MapReduce job.
2. Reduced memory requirements and processing time in the reducer by pre-aggregating the output of the mapper.

The reduce function would not change significantly when a combiner is used. The only difference is that the input to the reduce function would be the output of the combiner instead of the output of the map function.

▼ Hide question 5 feedback

you need to describe the actual changes in the function -2

more changes need to made instead of the different input.

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.).

Given M = 101 . But using 100 blocks for loading at once.
Purchase table = 1000 blocks
Product table = 2000 blocks

Sort merge is a 2 pass algorithm. First sorting both the tables,

Pass 1:

Sort Purchase table => 10 runs, 100 blocks/run - Read and Write purchase table so
cost = 2B(Purchase)

sort Product table => 20 runs , 100 blocks/run - Read and Write product table so
cost = 2B(Product)

B(Product) +B(Purchase) < M-1
Here the runs (10 +20 = 30) are less than M-1 (100)
So, extra step is not required.

Pass 1 : 2 * B(Purchase) + 2 * B(Product)

Pass 2 : Merging Phase

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

Given M = 101 .

Assumption: 100 pages are used for the sorting phase
Purchase table = B(Purchase) = 5000 blocks
Product table = B(Product) = 20,000 blocks

Pass 1:

Sort Purchase table => 50 runs, 100 blocks/run - Read and Write purchase table so
cost = 2 * B(Purchase)

Sort Product table => 200 runs , 100 blocks/run - Read and Write product table so
cost = 2 * B(Product)

We need to perform extra step as we have runs (200 +50 = 250) greater than M-1 (100).
B(Purchase) + B(Product) > M-1

So the extra step is for Product where we are merging 200 runs => 200/100 => 2 runs - Read and Write Product thus, for this extra step,
cost = 2B(Product)

Pass 2:

Merging Phase
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*20,000
= 1,15,000 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=Tr
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(product) AS num_products
FROM Purchase
GROUP BY buyer
ORDER BY num_products DESC
LIMIT 1;

ii. [3 points] DataFrame

```
purchase.groupBy('buyer').agg(fc.count('product').alias('num_products')).orderBy(fc.desc('num_products')).limit(1).select('buyer').show()
```

Question 11 6 / 6 points

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

i. [3 points] SQL

```
SELECT DISTINCT buyer
FROM Purchase
WHERE store = 'store1' AND buyer NOT IN (
  SELECT DISTINCT buyer
  FROM Purchase
  WHERE store = 'store2');
```

ii. [3 points] DataFrame

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

buyers = purchase1.select('buyer').subtract(purchase2.select('buyer')).distinct().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 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. [4 points] DataFrame

purchase.join(product, purchase.product == product.name).filter(purchase.store == 'store1').groupBy(product.category).agg(fc.avg(purchase.price).alias('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(DISTINCT Purchase.product) AS num_products
FROM Purchase
JOIN Person ON Purchase.buyer = Person.name
WHERE Purchase.store = 'store1' AND Person.city = 'LA';

ii. [3 points] DataFrame

purchase.join(person, purchase.buyer == person.name).filter((purchase.store == 'store1') & (person.city == 'LA')).agg(fc.count('*').alias('product_count')).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

```
personRDD.filter(lambda row: '123' in row["phone"]).map(lambda row: row["name"]).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
ORDER BY COUNT(DISTINCT product)
LIMIT 1;
```

ii. [3 points] RDD

```
purchaseRDD.map(lambda row: (row["buyer"], 1)).reduceByKey(lambda U, x: U + x).sortBy(lambda kv: kv[1]).take(1)
```

▼ Hide question 15 feedback

result is a tuple

Question 16 — 4 / 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 IN ('store1', 'store2')
GROUP BY buyer
HAVING COUNT(DISTINCT store) = 2;
```

ii. [3 points] RDD

```
buyers_store1 = purchaseRDD.filter(lambda row: row["store"] == 'store1').map(lambda row: row["buyer"])
buyers_store2 = purchaseRDD.filter(lambda row: row["store"] == 'store2').map(lambda row: row["buyer"])

buyers_store1.intersection(buyers_store2).distinct().collect()
```

▼ Hide question 16 feedback

WHERE store IN ('store1', 'store2')

The case could be some one bought product in store 1 and store 3

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 pr.category, avg(p.price) AS average_price
FROM Purchase p
JOIN Product pr ON p.product = pr.name
WHERE p.store = 'store1'
GROUP BY pr.category;
```

ii. [3 points] RDD

```
product_RDD = productRDD.map(lambda row: (row["name"], row))

purchase_RDD = purchaseRDD.map(lambda row: (row["product"], row))

join_RDD = purchase_RDD.join(product_RDD)

join_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 num_products
FROM Purchase p
```

```
JOIN Person pr ON p.buyer = pr.name
WHERE (p.store = 'store1' OR p.store = 'store2') AND pr.city = 'LA';
```

ii. [3 points] RDD

```
person_RDD = personRDD.map(lambda row: (row["name"], row))

purchase_RDD = purchaseRDD.map(lambda row: (row["buyer"], row))

join_RDD = purchase_RDD.join(person_RDD)

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

Done