Total points: 200

Part 1: Spark Batch processing - 120 points

Problem 1A: (Points: 30) Local Spark setup

1. Input data format: <timestamp> <URL> <userID>
2. generate 2-5 files, with 1k-2k events each, in the specified format, with timestamps that span 3-4 months in time
3. write 3 Spark jobs, that process these generated files and answer the "hourly" aggregates queries:

Query 1: get count of unique URLs by hour

Query 2: get count of unique visitors per URL by hour

Query 3: get count of unique (by userId) clicks per URL by hour

4. compare DAGs, jobs and RDDs of the three jobs, explain the differences

The Jobs need to use the RDD Interface only .

Please **do not** use dataframe or Dataset API's .

You can run your jobs locally (from Eclipse/Notebooks) or via spark-shell .

Problem 1B - Spark on EMR setup

1. Place your input data into HDFS
2. Modify your jobs to use data from HDFS
3. Run your jobs on an EMR cluster with Spark installed
4. compare DAGs, jobs and RDDs of the jobs run locally vs on AWS cluster, explain the differences

Problem 2: (Points: 30)

1. Modify your generated data to also contain UUID for each event: <uuid> <timestamp> <URL> <userID>; generate 2-5 files , with 1k-2k events in this format, also spanning 3-4 months in time; place your input data into a S3 bucket and create a folder ; lets call it Dataset1
2. Make sure you have at least 20 unique userIDs
3. Duplicate your input files - create a copy of each file, with a different name. We are trying to simulate having "duplicate" events, with the same UUID. In

our case it is an extreme - each event will have a duplicate event with the same UUID; Let call it Dataset 2. Place these files also in S3 in the same bucket but under a different folder.
4. Modify your Spark jobs to de-duplicate the data and execute the same queries
5. Run your jobs on Dataset1 first
6. Run your jobs on both Dataset1 and Dataset2 - compare the results with those from Step 4, they should be the same because you have added the de-dup logic
7. Run these jobs on an EMR cluster with Spark
8. compare DAGs, jobs and RDDs of the jobs run locally vs on AWS cluster, explain the differences

Problem 3: (Points: 30)

1. Define an Avro schema for your data (with UUIDs); explain your schema design
2. Write a Spark Job to read the input data (both Dataset1 and Dataset2 you generated for Problem 2) from S3 and convert it to Avro format (DatasetAvro) - and write the results back into S3 in a different folder.
3. Modify your Problem 2 Spark jobs to read input data in the Avro format, de-dupe the events and give answers to the same 3 queries
4. Write a Spark job that reads DatasetAvro, de-dupes this data and writes it back to S3 but in a Parquet format. Call it DatasetParquet
5. Run the jobs on an EMR cluster with Spark
6. compare DAGs, jobs and RDDs of the three jobs, explain the differences

Problem 4: (Points: 30)

1. Generate a new dataset (DatasetCommunity) in the format: <userID> <communityID>; make sure that you have a few userIDs per each "communityID"; use the same 20 userIDs you have generated for the Dataset1 in Problem2
2. For example:

user01 community01

user02 community01

user03 community02

user04 community02

...

3. Write a new Spark job that loads both DatasetParquet and DatasetCommunity

4. Implement the following query:

get counts of clicks per URL per communityID

5. Run the jobs on an EMR cluster with Spark

6. compare DAGs, jobs and RDDs of this job with those from Problem 2, explain the differences

Problem 5: Bonus: 30 points

1. Take/modify the generated data/files you produced for Problem1, make sure you have at least 100 K records spread over 10-20 files; the Input data format is <timestamp> <URL> <userID>
2. You want to have at least 3K records or more for the 3 URL's and 3 ID's below , we will focus on these 3 URL's only for this problem
    1. <timestamp> <www.cnn.com (Links to an external site.)Links to an external site.> <cnnssoadm>
    2. <timestamp> <www.angieslist.com (Links to an external site.)Links to an external site.> <aglssoadm>
    3. <timestamp> <www.nytimes.com (Links to an external site.)Links to an external site.> <nytssoadm>
3. Task 1: modify your Spark job from Problem 1 to calculate hourly counts of events (number of records) for the 3 specified target URLs only; the URLs can be passed as your job arguments via main() method, or hardcoded in a list in your code. Note the job execution time
4. Task 2: modify your job to use "broadcast variables" to specify the target 3 URLS. Note the job execution time
5. Report the time differences between #4 and #5;

You can run the jobs on either EMR Spark Cluster or locally (make sure to have local[*] so all cores are used)

Problem 6: Bonus: 30 points

1. Take/modify the generated data/files you produced for Problem1, make sure you have at least 1 Mil records spread over 20 files; the Input data format is <timestamp> <URL> <userID>
2. Modify your Problem 1 Spark jobs to use transformations specified below:

2a - use **groupbykey** , record the time it takes to run the same 3 queries as in Problem 1

2b - use **reducebykey instead** , record the time it takes to run the same 3 queries as in Problem 1

3. Next for both 2a and 2b modify the code to force the “number of partitions” to be 200 - record the time to execute the same 3 queries

4. Report the time differences between #2 (a and b) and #3

Part 2: Batch Views - 80 points

Problem 7: (Points: 50) Batch Views with NoSQL DBs (MongoDB)

1. Install MongoDB locally (or on Docker)
2. design and create tables to store whatever information is needed to calculate the following reports [most likely you will need to store the results of your batch views, computed in Problem 1, and some additional information] :

Report 0: the same 3 original queries (hourly counts)

Report 1: Daily count of unique URLs

Report 2: Daily count of unique visitors

1. Modify your Problem1 Spark job to store required data into the new tables in the Mongo DB
2. write and execute Mongo DB queries/aggregates to generate the specified reports

Problem 8: (Points: 30) Batch Views with RDBMS

1. Install a local MariaDB

2. Do the same tasks as in Problem 7, with the goal of running the same reports, but using the MariaDB this time - you have to modify your Spark job to write the results to MariaDB
3. explain the differences in table designs and aggregation/reporting views calculation approaches between MongoDB and MariaDB
4. *[Bonus: +20]* repeat the same , running your Spark job on an AWS Spark cluster and having your MariaDB running on a separate EC2 instance

Problem 9: Bonus: (Points: +20)

1. given the same input dataset as in Problem1, decide what would be the simplest Batch View to be calculated and stored, in order to run the following type of reports/queries:
    1. what is the busiest hour of the day for a specified URL for a specified day? [URL and the 'day date' are input parameters for the query]
    2. "busiest hour" is defined as the hour that has the most events for that URL (total count, not uniques)
2. use can choose which DB to use (MongoDB or MariaDB)
3. modify your Spark job from Problem1 to calculate an store the identified required information into the DB
4. run the specified query for some URL and some day (you pick, depending on what dates/urls you have in your input data)

Problem 10: Bonus: (Points: +50) Spark + Dynamo DB on AWS

1. Instead of using MariaDB or MongoDB, use Dynamo DB on AWS for your Batch Views storage
2. Define appropriate hash key, range key and necessary attributes to compute the same reports
3. Find an approach to calculate the same Reports as in Problem 7 and 8
4. explain what you could and could not achieve with DynamoDB (easily) as opposed to MongoDB and MariaDB, and why

Grading Rubrik

Total Points: 200