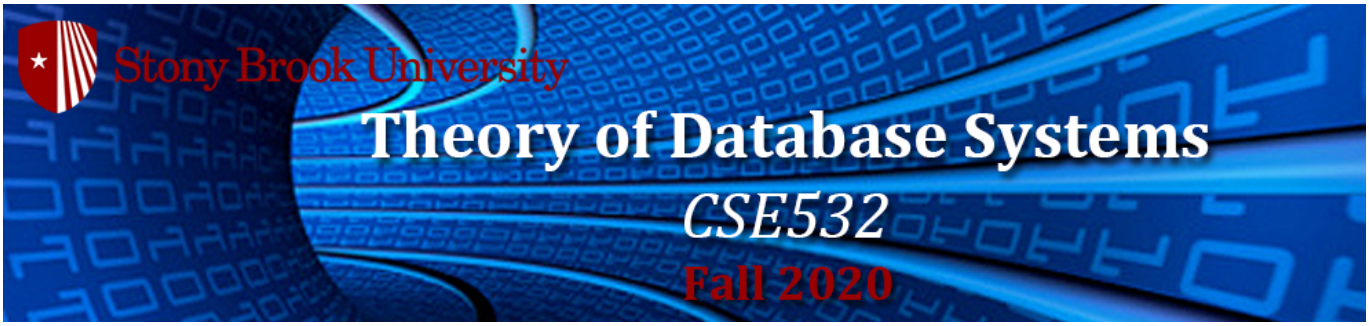

Home

Home
Syllabus
Schedule
Homework
Textbooks
Examples
Instructor
Resources

Homework >

Homework4hadoop

Please start early. Although it may seem trivial, the programming aspect might be time sensitive and especially debugging is **NOT** easy.
Total credit: 25; extra credit: 10

Pseudo-Distributed Hadoop And Spark

This is just to warm you up. In reality, Hadoop and Spark are usually deployed over actual distributed cluster nodes. However, for the purpose of this course and to get an idea of the environment, we're starting with a local single node Hadoop and Spark setup (HDP sandbox, Cloudera quickstart VM or Docker).

Even in real world, it is always a good idea to start small and build - Keep It Simple, Stupid (KISS Principle)

Array of Things (AoT) Data Analysis

The Array of Things (AoT) is an experimental urban measurement project comprising a network of interactive, modular devices, or "nodes," that are installed around Chicago to collect real-time data on the city's environment, infrastructure, and activity. These measurements are published as open data for research and public use. AoT essentially serves as a "fitness tracker" for the city, measuring factors that impact livability in Chicago such as climate, air quality and noise.

The project was funded by the U.S. National Science Foundation and by late 2019 roughly 130 nodes had been installed throughout the city, typically at street intersections, 22-24 feet above the street. The project plan is to deploy roughly 150 nodes by mid-2020, involving both replacing older nodes (some of which were installed as early as 2016) and adding locations.

As of today the nodes measure temperature, barometric pressure, light, vibration, partiulate matter (PM2.5), carbon monoxide, nitrogen dioxide, sulfur dioxide, ozone, and ambient noise levels. A novel aspect of AoT nodes is that they can analyze images and sound without saving or transmitting the data.

Task Requirements

(1) [5 points] Building on the simple WordCount example done in class and Hadoop tutorial, your task is to perform simple processing on provided Array of Things dataset.

- The first task is to count the **total number of values/records reported for each type of parameter** in the dataset. For example, parameter types can be (but NOT limited to) `temperature, car_total, person_total, bins` etc.
- Name your program `AoT_1.java`
- Program arguments description
  - The HDFS path to your input data file
  - The HDFS output path for your program. (NOTE: You should remove the output path after every execution of your program. Hadoop cannot start a job if output directory is already created)

(2) [10 points] General analysis usually require processing in multiple modes. For instance, analysis can be performed on a particular type of parameter and on values inside a given date range only. Similarly, we can perform aggregations on a particular field in the dataset.

- Your second task is to modify your program to perform user defined aggregations on a subset of data inside user defined date ranges
- Name your program `AoT_2.java`

- Note that you'll have to use `timestamp` field in the dataset to filter records for given date range
- Note that you **DON'T** have to perform error handling for invalid date format. Assume that the dates given in parameters will always be in correct defined format
- Also, your result should contain data including start and end dates

- Program arguments description

  - The HDFS path to your input data
  - Start date (YYYY-MM-DD)
  - End date (YYYY-MM-DD)
  - The name of the parameter e.g. temperature etc. Only one parameter can be specified at a time
  - The aggregate operator i.e. **avg**, **min**, and **max**
  - The HDFS output path for your program. (NOTE: You should remove the output path after every execution of your program. Hadoop cannot start a job if output directory is already created)

(3) [5 + 5 points] Implement task-1 and task-2 for Spark

- Name your programs `SparkAoT_1.{java,py,scala}` and `SparkAoT_2.{java,py,scala}` based on your language of implementation

Dataset

Dataset Description

Original AoT dataset contains several levels of information (AoT Dataset Description). Extensive parsing and cleaning is required to extract information of interest. To focus on Hadoop/Spark working instead of spending your energies on data parsing, we provide you with abridged version of the dataset.

The dataset is in json format where each line represents the following fields

```
{
   "features":{
      "parameter":"car_total",
      "sensor":"image_detector",
      "node_id":"image",
      "value_hrf":0
   },
   "latitude":41.692703,
   "geohash":"dp3tnjumd70h",
   "id":"",
   "timestamp":1592020801000,
   "longitude":-87.62102
}
```

Dataset Download

You can download sample dataset from the shared google drive.

The AWS (bigger) dataset can be accessed using the following path. You don't have to download or upload it. It's already uploaded to an s3 bucket which you can give as an input to your code.

```
s3://cse532/aot_data_big.json
```

Examples

To execute Hadoop jobs using Java API (Terminal Commands)

```
hadoop AoT.jar AoT_1 /cse532/input/aot-small-10k.json /cse532/output/

hadoop AoT.jar AoT_2 /cse532/input/aot-small-100k.json 2020-01-01 2020-03-31 temperature avg /cse532/output/
```

To view output (Terminal Commands)

(list contents of HDFS output directory)

```
hdfs dfs -ls /cse532/output/
```

(print out the contents of output files to terminal)

```
hdfs dfs -cat /cse532/output/part-*
```

To execute Spark jobs using Java API (Terminal Commands)

```
spark-submit --class AoT_1 SparkAoT.jar /cse532/input/aot-small-10k.json /cse532/output/
```

```
spark-submit --class AoT_1 --master local[2] SparkAoT.jar /cse532/input/aot-small-10k.json /cse532/output/

spark-submit --class AoT_2 SparkAoT.jar /cse532/input/aot-small-100k.json 2020-01-01 2020-03-
31 temperature min /cse532/output/
```

To execute Spark jobs using Python API (Terminal Commands)

```
spark-submit SparkAoT_1.py /cse532/input/aot-small-10k.json /cse532/output/

spark-submit --master local[2] SparkAoT_1.py /cse532/input/aot-small-10k.json /cse532/output/

spark-submit SparkAoT_2.py /cse532/input/aot-small-100k.json 2020-01-01 2020-03-
31 temperature max /cse532/output/
```

Submission

1. Submit **AoT_1.java** for task-1 (It should contain the map and reduce functions)
2. Submit **AoT_2.java** for task-2 (It should contain the map and reduce functions)
3. Submit **SparkAoT_1.{java, py, scala]** and **SparkAoT_2.[java, py, scala]** for task-3

[Extra Credit] Amazon Elastic MapReduce (EMR)

In real world, distributed systems such as Hadoop and Spark come in handy when you have to process huge amounts of data. Processing such data on local machines is prohibitive unless you are have access to Super Computers. The good news, however, is that the code written for pseudo distributed mode can safely be considered deployable on actual distributed cluster without much modifications.

Task Requirements [10 **points**]

Real world big datasets come and in all shapes and sizes and can hardly be considered clean out-of-the-box for useful analysis. An important initial phase for any dataset before using it for actual analysis is to clean it i.e. remove empty values, remove outliers etc.

In this task, you'll work on a considerably larger dataset i.e. ~1.5 GB in size in the same format as described above. You will need to process this dataset in an actual distributed cluster. We'll be using Amazon EMR for this purpose where a lot of deployment and management details have been abstracted for you.

- For this task, you can write your code for Hadoop **OR** Spark whichever you feel comfortable with
- The goal of this task is to remove outlier values i.e. `value_hrf` (hrf: human readable format) from the dataset. In addition to removing the outliers, we would also like to get the location `(latitude, longitude)` and `geohash` of the node that reported faulty `value_hrf`. So essentially your output will look like

```
node,(lat,long),geohash [value_hrf1, value_hrf2 ...]
node,(lat,long),geohash [value_hrf3, value_hrf4 ...]
....
```

-
- Program arguments description
  - The S3 or HDFS path to your input dataset
  - Parameter name to check for outliers e.g. `temperature`
  - Min value for specified parameter's `value_hrf`
  - Max value for specified parameter's `value_hrf`
  - The S3 or HDFS output path

Amazon EMR

ALWAYS TERMINATE YOUR CLUSTER (CHECK/TURN OFF TERMINATE PROTECTION AND USE TERMINATE CLUSTER AFTER STEPS)

Documentation on how to run a custom JAR using the GUI interface

http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-launch-custom-jar-cli.html
You can use the AWS Command Line Interface to run the JAR, but it will require you to read the documentation (not recommended due to the length and the difficulty of setting up security keys.)

You can view the screenshots of the steps to set up a cluster below in the download section of this page.

Important Setup Instruction
In Management Console, select Storage -> Amazon S3.

First in Amazon S3, create a bucket for yourself.
Create an input subdirectory and a program subdirectory in your bucket.
Upload your data (programs and input) into the corresponding folders.

Second, in Amazon EMR, use advanced option.
Select only Hadoop or Spark to be installed. Delete Hive, PIG and etc.

Recommended settings
1) Use 2-4 nodes to test your data.
2) Leave auto-terminate on.

To add steps
For both JAR and streaming execution, treate Amazon S3 path with prefix `s3://`, as if the data comes the distributed file system (HDFS). S3 simulates HDFS to Hadoop EMR.

AWS input S3 location (used as input path)

```
s3://bucket_name/aot.json
```

You can also upload the files to your own customized S3 location, but that is not recommended.

AWS output S3 location:
Use a directory that doesn't exist on S3. Otherwise, your job will fail. If you would like to reuse the output directory, please delete the directory from S3 console.

It is highly recommended that you test your program locally on your VM before testing it on the bigger data set.

Submission

1. Submit **SparkAoTExtra.{java,py,scala}** (Depending on your implementation language)
2. Submit **part-000*** with the output result containing node information for outlier values for any one of the parameter

step1.PNG (207k)	Furqan Baig, Nov 4, 2020, 12:11 AM	v.1
step2.PNG (190k)	Furqan Baig, Nov 4, 2020, 12:11 AM	v.1
step3.PNG (150k)	Furqan Baig, Nov 4, 2020, 12:11 AM	v.1
step4.PNG (220k)	Furqan Baig, Nov 4, 2020, 12:11 AM	v.1
step5.PNG (222k)	Furqan Baig, Nov 4, 2020, 12:12 AM	v.1
step6.PNG (116k)	Furqan Baig, Nov 4, 2020, 12:12 AM	v.1
step7.PNG (127k)	Furqan Baig, Nov 4, 2020, 12:12 AM	v.1
step8.PNG (161k)	Furqan Baig, Nov 4, 2020, 12:12 AM	v.1

Comments

You do not have permission to add comments.