Methods of Cloud Computing

Winter Term 2019/2020

Practical Assignment No. 4

Due: 30.01.2020 23:59

The primary goal of this assignment is to get familiar with data-flow engines at the example of Apache Flink. You will setup and evaluate two small applications locally and on a distributed Flink installation.

1. Prepare and Test a Local Flink Setup

Download and install Apache Flink on your test system. Flink requires a working installation of Java 8, thus you might want to consider downgrading your current Java installation.

Test your Flink setup by running the `SocketWindowWordCount` example.

Adapt the Word Count example to ignore the case of all words, as well as any non-alphabetical characters. Do not employ time windows, rather process the entire input. The input and output files' names are passed via command line arguments. You are free to solve this task in Java or Scala. The output must be a CSV (comma separated values) file as shown below.

```
word,occurrences
lorem,5
ipsum,3
dolor,8
...
```

Hints:

- The header row in the output CSV file is optional.
- It is okay to output a rolling word-count. Only printing the final count will give you +5 extra points.
- For your convenience we supplied you a sample text on the lecture website (`tolstoy-war-and-peace.zip`).

Outputs:

- `WordCount.[java|scala]`

2. K-Means on Mobile Cellular Datasets

Write a Fink program, that clusters cellular tower data from OpenCellId.org. Base your code on the Flink k-means batch example (Scala or Java).

Your program should fulfill the following requirements:

- Create clusters of GSM and UMTS towers (ignore LTE towers)
- Allow an optional filter for particular network operators given by a comma-separated list of Mobile Network Codes (abbreviated as `--mnc`)
- The number k of clusters should be configurable via a parameter, and if started without the parameter, use the number of LTE towers as default value
- The initial placement of the cluster centroids should be the positions of the LTE towers (pick the first k towers, if there are less clusters than towers).
- Your program should consist of a single source code file and output a CSV file of the GSM and UMTS coordinates in the format below. The first column is the centroid id (you can use the LTE tower id if LTE towers are used as initial positions). The two following columns are the GSM/UMTS cell coordinates.

```
centroid,lon,lat
26970112,13.34986,52.497252
26970112,13.349873,52.497557
5226755,13.285064,52.524
...
```

An example call clustering Telekom towers in Berlin around 500 LTE towers might look like:

```
flink run CellCluster.jar --input $PWD/berlin.csv \
                          --iterations 10 \
                          --mnc 1,6,78 \
                          --k 500 \
                          --output $PWD/clusters.csv
```

We provide you database dump of all cell towers in Berlin and Germany on the lecture website (`opencellid_data.zip`).

Hints:

- You can plot your output data with the provided `plot_clusters.py` script
- We provide you with a database dump of all cell towers in Berlin and Germany on the lecture website (`opencellid_data.zip`). Test your program with these datasets, using different configurations. Start with a low number of clusters and iterations and gradually increase these parameters to see what your system can handle.
- The CSV header row in the output file is again optional

Outputs:

- `CellCluster.[java|scala]`

3. Dead Spots in Cellular Datasets

Write a Flink batch program in Scala or Java, that determines the cellular coverage for a given list of points. For this task, coverage defined as "this point is in range of a cellular base station of the giving generation."

Your program should fulfill the following requirements:

- Work on the same input data from OpenCellId passed with `--input`
- Allow for filtering by Mobile Network Code (abbreviated as `--mnc`)
- Expect a CSV file of cartesian points (longitude, latitude) for with the coverage should be tested (`--spots`)
- Your program should consist of a single source code file and output a CSV file, where each line stands for a tested spot with its longitude, latitude and coverage for each mobile network generation (GSM, UMTS, LTE)

```
lon,lat,GSM,UMTS,LTE
13.131675,52.393853,1,1,1
14.359243,51.939038,1,1,0
...
```

An example call testing Telekom coverage in Brandenburg/Berlin might look like:

```
flink run DeadSpots.jar --input $PWD/germany.csv \
                        --spots $PWD/testspots.csv \
                        --mnc 1,6,78 \
                        --output $PWD/deadspots.csv
```

Hints:

- Measuring the distance in meters between cartesian coordinates is hard. For this task you can assume the earth the be a sphere and use the Haversine formula.
- Think about data distribution and partitioning. Which datasets will have the potential higher element count for which kind of experiments?
- Provide Flink with hints when using expensive operations (joins, products, sorting or groups).
- The CSV header row in the output file is again optional

Outputs:

- `DeadSpots.[java|scala]`

4. Use a Distributed Flink Installation

Execute your programs on a distributed Flink installation. Choose one of the below options for this.

1. **Amazon EMR**
   - Study the Amazon EMR online documentation
   - Use S3 buckets to provide storage for your input and output files
   - This option may not be applicable, if you only have an AWS Education Starter account
   - When using this option, submit your configuration files for the AWS CLI and/or Flink cluster.
2. **Flink in Kubernetes**
   - Use the Kubernetes from the previous assignment to deploy Flink and Hadoop on at least 3 worker nodes
   - Use the Flink *session cluster* installation (NOT Flink job cluster).
3. **Flink in VMs**
   - As an alternative to your Kubernetes cluster, you can manually install Flink and Hadoop on three VMs and configure the installation to work together as a cluster.

Hints:

- For your Flink in Kubernetes/VMs installation, use HDFS as data input/output. Change the URLs for your files to something like: `hdfs:///path/to/data.csv`

Outputs:

- `listing.txt`
  - Containing all commands you used for preparing your Flink cluster, including comments explaining what the commands do
- `configurations.json, flink-conf.yaml` (Only for Amazon EMR)

5. Experiments & Discussion

Run your programs with different job configurations:

- Different numbers of iterations or clusters for k-means
- Different initial placements

Write a few sentences in the file `discussion.txt` about the configurations you have tried and what you have observed regarding the run time and results of the jobs.

Answer the following questions separately for all three programs:

- Which steps in your program require communication and synchronization between your workers?
- What resources are your programs bound by? Memory? CPU? Network? Disk?
- How could you improve the partitioning of your data to yield better run-time?

Outputs:

- `discussion.txt`

6. Submission Deliverables

Use the OpenSubmit system running at https://www.dcl.hpi.uni-potsdam.de/submit/ to submit your solutions. Every group member must register on the OpenSubmit platform, but only one member should submit the final solution. **Select your co-authors when creating a new submission.** Your solution will be validated automatically, you can resubmit your solution multiple times until it passes the tests.

Expected contents of your solution zip file:

- `WordCount.[java|scala]`
  - **1 source file, 10 points [+ 5 extra points]**
- `CellCluster.[java|scala]`
  - **1 source file, 25 points**
- `DeadSpots.[java|scala]`
  - **1 source file, 30 points**
- `listing.txt, [configurations.json, flink-conf.yaml]`
  - **1 listing file, 2 config files for Amazon EMR, 15 points**
- `discussion.txt`
  - **1 text file, 20 points**

Total points: 100 [+ 5 extra points]