E6Data Coding Challenge

Introduction

E6Data is an OLAP Query Processing Engine built for Open Lakehouse. This coding challenge will give you a glimpse of the kind of problems you will be encountering on a frequent basis and will help us understand how you approach and solve such problems.

Problem Statement

Implement a distributed text processing engine. It should be able to read a set of .txt files present in a directory which contains textual data and answer a fixed set of questions which the user selects. The engine should be able to to utilize processing power of multiple cores available on one single compute node as well as scale to multiple compute nodes. There will be a driver program which will talk to the engine instances for sending user requests and displaying the results. Following are the details of various components you need to build.

Driver Program

This program takes only one argument: `input.txt`. The format of this file is:

- *** Line 1:** Fully qualified path to the directory which contains the text files.
- *** Line 2:** Space separated numbers representing the following commands:
    - total file count (1)
    - total word count (2)
    - total distinct word count (3)
    - top 100 words with frequencies in descending order of frequency of occurence (4)

The driver should print the output to the console where each line contains the output of each command. The output format for each command's result is:

Command 1:

```
Unset
file_count time_taken_to_execute_command_in_millis
```

Command 2:

```
Unset
total_word_count time_taken_to_execute_command_in_millis
```

Command 3:

```
Unset
total_distinct_word_count time_taken_to_execute_command_in_millis
```

Command 4:

```
Unset
word1 frequency1 word2 frequency2 ... word100 frequency100
time_taken_to_execute_command_in_millis
```

There will be at least 100 unique words across all files.
For example:
input.txt:

```
Unset
/home/john/text_files
1
2
3
4
```

output on the console:

```
Unset
20 200
40000 3000
7000 4200
a 600 an 550 is 400 was 350 ... remaining words ... joe 20 sam 10 10000
```

Engine Instance

Each engine instance is capable of executing text processing tasks. It is upto you to come with what these text processing tasks are. For example, a task could be reading exactly one file and counting words. You can decide how to break user commands into smaller tasks and process them. The engine instances accepts tasks from the driver, executes them and returns the output back to the driver. The communication protocol and messaging format between the driver and the engine instances is also left to you.

Shell script (run.sh / run.py)

To run the driver and the engine, you should provide a shell or python script which will do the following:
1. Start 3 instances of engines.
2. Start the diver program with the input.txt supplied to the script.

For example:

```
Unset
$ cd project_dir
$ ./run.sh input.txt
20 200
40000 3000
7000 4200
a 600 an 550 is 400 was 350 ... remaining words ... joe 20 sam 10 10000
```

Things to Note

- You can use your language of choice for implementing the engine and the driver.
- You should zip you project and name it in this way: \<bits_id\>_\<full_name\>.zip
- The zip should include the following:
    - source code
    - compiled artifacts/binaries for Linux and Intel Architecture.
    - - shell script which will be run from your project directory. Make sure the paths to the binaries referred in it are w.r.t. the project directory.
    - - A ReadMe.txt with the following informations:
    - - Your project's architecture.
    - - Source code layout
    - - Instructions to run the script
    - - Instructions to compile in case we face any issue.

Evaluation

Your program will be run on an Intel machine with Linux OS with 32GB RAM and 8 cores. Please ensure that your binaries are compiled for the same. In case they are not, please provide instructions to build your project. In absence of the same, we won't be able to proceed with the following evaluation. All three instances of the engine will start-up on the same machine. You can use different port ranges on the machine to avoid clashes.

Evaluation will happen in the following order.

- Run shell script which accepts input.txt as the only argument.
- Executes the commands in input.txt and prints the output on the console.
- Time taken to run the full script.

Tips for building and running the Project

Java

- Provide your project as a maven or gradle project. You can use Eclipse or IntelliJ (preferred) as your IDE which can help you with generating a maven or gradle project. In case not, then provide a script to build the project.
- Ensure that it can build and run on Java 8 or above.
- The engine and the driver program should can be built as fat jar to include all dependencies.

Python

- Provide the requirements.txt file so that we can install dependencies in case we are not able to run your project directly due to missing packages. Python 3.x will be the installed version on our systems.

C/C++

- Please provide your code as a make or a cmake project in case we need to compile.
- Provide a list of dependencies and commands to install them on Linux. Also, mention your Linux flavour (Ubuntu/Debian/Fedora/CentOS/OpenSUSE/Redhat etc.)

- Pre compiled static binaries are preferred, so that we don't have to install dependencies.

Selection criteria

- Your code should at the minimum compile on the Linux machine. (5 pts)
- If it compiles, it should run without any error. (20 pts. -> 5 pt. for each command)
- If it runs, it should provide the output as quick as possible. (20 pts. -> 5 pt. for each command)
- If it provides the output, then it will be checked for code quality. In this we will check the project structure, variable naming, modularization and how well documented it is. (10 pts.)

You will be ranked based on the points you score.

Please fill the google form once you are done

https://forms.gle/x1ABrRKabtivyNNw9