Branch: master ▾ Find file Copy path

CS167 / Labs / Lab2 / CS167-Lab2.md

aseldawy Update CS167-Lab2.md

da409a4 on Apr 12

2 contributors

Raw Blame History

117 lines (91 sloc) 7.83 KB

Lab 2

Objectives

- Understand the differences between the running modes of Hadoop.
- Setup the Hadoop Distributed File System on your local development machine.
- Compare the performance of HDFS to the local file system.

Prerequisites

- Follow the instructions in Lab #1 to setup the development environment.
- Download this Sample File and decompress it. We will use it for testing.

Lab Work

Follow the instructions below to complete this lab. If you have any questions, please contact the TA in your lab. Make sure to answer any questions marked by the (Q) sign and submit the deliverables marked by the (S) sign.

I. Setup (10 minutes)

1. Create a new Java project using Maven for Lab2.
2. Import your project to IntelliJ IDEA.

3. In `pom.xml` file, add dependencies on `org.apache.hadoop:hadoop-common` and `org.apache.hadoop: hadoop-hdfs`

II. Main Program (45 minutes)

Write a main function with the following specifications.

1. It should take two command-line arguments; the first for the *input* file and the second for the *output* file.
2. If the number of command line arguments is incorrect, write a reasonable error message and *exit*.
3. Store the two arguments in local variables of type `org.apache.hadoop.fs.Path`
4. Retrieve the correct file system for the two files and store in a variable of type `org.apache.hadoop.fs.FileSystem` . *Hint, use the function Path#getFileSystem(Configuration)*
5. Check whether the input file exists or not. If it does not exist, write an error message and *exit*.
6. Similarly, check whether the output file exists or not. If it already exists, write an error message and exit.
7. Use FileSystem API to open the input file and copy all its contents to the output file. Measure the total time that it takes to do this step. *Hint: Use the method System#nanoTime()*
8. Write a message similar to the following to the standard output.

```
Copied 1255 bytes from 'README' to 'README_copy' in 0.016095 seconds
```

The message should show the correct size, input, output, and total time according to how the program ran.

III. Benchmark Local File System performance (10 minutes)

Now, it is time to use your program to benchmark the performance of the local file system and HDFS.

1. Compile your program into a runnable JAR file.
2. Test your program on a small file, e.g., a `README` file to make sure that it works correctly.
3. Now, test it on the sample file that you downloaded, `AREAWATER.csv` . (Q1) Verify the file size and record the running time.

4. Make a copy of the file using your file system command and record the time, e.g., using `time` command on Linux, or using `Measure-Command { ... }` command on Windows PowerShell. (Q2) Record the running time of the copy command.
5. (Q3) How do the two numbers compare? (The running times of copying the file through your program and the operating system.) Explain *in your own words* why you see these results.

IV. Configure and Run HDFS (20 minutes)

To run HDFS, you need at least one name node and one data node.

1. Configure Hadoop to use HDFS as the default filesystem. Add the following lines to `$HADOOP_HOME/etc/hadoop/core-site.xml`

```
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
```

2. Initialize the files for HDFS by running the command `hadoop namenode -format`
3. Start the master node by running `hadoop namenode`
4. Start the data node by running `hadoop datanode`

Note: You will need to run the name node and data node in two separate windows. Keep them open so you can see their progress as your program runs.

V. Use the Command-Line Interface (CLI) to Access HDFS (20 minutes)

1. List the contents in HDFS under the root directory. `hdfs dfs -ls /`
2. Create a home directory for yourself if it does not exist. `hdfs dfs -mkdir -p .`
3. Upload a small file, e.g., a `README` file to your home directory in HDFS. `hdfs dfs -put <filename>`
4. List the files in your home directory. `hdfs dfs -ls`
5. List all available commands. `hdfs dfs`

VI. Run Your Program with HDFS (20 minutes)

1. Run your program again from the command line to copy the file. (Q4) Does the program run after you change the default file system to HDFS? What is the

error message, if any, that you get?
2. Run your program again, this time specify the full path to your local file (both input and output) and explicitly specify the local file system (`file://`)
3. (Q5) Use your program to test the following cases and record the running time for each case.
   i. Copy a file from local file system to HDFS
   ii. Copy a file from HDFS to local file system.
   iii. Copy a file from HDFS to HDFS.

Note: to explicitly specify the HDFS file system, use the scheme `hdfs://`

VII. Submission (15 minutes)

1. Add a `README.md` file and include all the answers to the questions above in the `README` file. Note: Don't forget to include your information in the README file.
2. Add a table that shows the running time for copying the test file in the five cases mentioned above.
3. Add a script `run.sh` or `run.ps1` that will compile your code and run the five cases on the input file `AREAWATER.csv`
4. (S) Submit your compressed file as the lab deliverable.

- Note 1: Don't forget to remove any unnecessary test or binary files.
- Note 2: In your `run` script, you can access the current working directory to specify the full path correctly when you specify the copy command. Assuming that the input file `AREAWATER.csv` is in the current working directory, you can run one of the following commands.

In a Linux shell script:

```
hadoop jar <JARFILE> file://`pwd`/AREAWATER.csv hdfs:///AREAWATER.csv
```

In a Windows PowerShell script:

```
hadoop jar <JARFILE> file:///$pwd/AREAWATER.csv hdfs:///AREAWATER.csv
```

VIII. Bonus Task (30 minutes) +3 points

1. Build a separate main class `AppB` that takes one input file. The main class should do the following steps.

2. Make sure that the file already exists.
3. Make 10,000 reads from the file at random positions. Each one should read 8,192 bytes. You can discard the bytes that you read immediately after they are read.
4. Measure the total time needed to do the 10,000 reads.
5. (Q6) Test your program on two files, one file stored on the local file system, and another file stored on HDFS. Compare the running times of both tasks. What do you observe?
6. Update your `run.sh` or `run.ps1` script to run the AppB class after the previous one. Since a JAR file cannot have two main classes, you will need to modify your running commands to explicitly specify the main class in each case. Additionally, you will need to remove the main class from your `pom.xml` configuration file.

Notes

- Make sure to follow the naming conventions that are mentioned in Lab #1.
- Failure to follow these instructions and conventions might result in losing some points. This includes, for example, adding unnecessary files in your compressed file, using different package names, using a different name for the compressed file, not including a runnable script, and not including a `README` file.