Assignments

Data-Intensive Distributed Computing (Fall 2023)

0 (assignment0-451.html) 1 (assignment1-451.html) 2 (assignment2-451.html) 3 (assignment3-451.html) 4 (assignment4-451.html)

5 (assignment5-451.html) 6 (assignment6-451.html) 7 (assignment7-451.html) Final Project (project-451.html)

Assignment 0: Warmup due 4:00 pm EST Sept. 20

The purpose of this assignment is to serve as a warmup exercise and a practice "dry run" for the submission procedures of subsequent assignments. You'll have to write a bit of code but this assignment is mostly about the "mechanics" of setting up your Hadoop development environment. In addition to running single-node Hadoop (also known as "local mode"), either on the Linux student CS environment or on your own machine, you'll also try running jobs on the Datasci cluster.

The general setup is as follows: you will complete your assignments and check everything into a private Git repo. Shortly after the assignment deadline, we'll pull your repo for marking.

Setting up Hadoop and Spark

Hadoop and Spark are already installed in the `linux.student.cs.uwaterloo.ca` environment (you just need to do some simple config). Alternatively, you may wish to install everything locally on your own machine. For both, see the software page (software.html) for more details.

Bespin is a library that contains reference implementations of "big data" algorithms in MapReduce and Spark. We'll be using it throughout this course. Go and run the Word Count in MapReduce and Spark (https://git.uwaterloo.ca/cs451/bespin) example as shown in the Bespin README (clone and build the repo, download the data files, run word count in both MapReduce in Spark, and verify output). Assuming you are using `linux.student.cs.uwaterloo.ca` (or if you have properly set up your local environment), this task should be as simple as copying and pasting commands from the Bespin README.

Time to write some code!

Create a **private** repo called `bigdata2023f` on UW GitLab (https://git.uwaterloo.ca/). If you're not already familiar with Git, there are plenty of good tutorials online: do a simple web search and find one you like.

What you're going to do now is to copy the MapReduce word count example into you own private repo. Start with this `pom.xml` (assignments/pom.xml): copy it into your `bigdata2023f` repo.

Next, copy:

- `bespin/src/main/java/io/bespin/java/mapreduce/wordcount/WordCountSimple.java` over to
- `bigdata2023f/src/main/java/ca/uwaterloo/cs451/a0/WordCountSimple.java` .

Tip: `mkdir -p` creates parent directories as needed. Open up this new version of `WordCountSimple.java` using a text editor (or your IDE of choice) and change the Java package to `ca.uwaterloo.cs451.a0` .

Now, in the `bigdata2023f/` base directory, you should be able to run Maven to build your package:

```
$ mvn clean package
```

Once the build succeeds, you should be able to run the word count demo program in your own repository:

```
$ hadoop jar target/assignments-1.0.jar ca.uwaterloo.cs451.a0.WordCountSimple \
   -input data/Shakespeare.txt -output wc
```

You should be running this in the Linux student CS environment or on your own machine. Note that you'll need to copy over the Shakespeare collection in `data/` . The output should be exactly the same as the same program in Bespin, but the difference here is that the code is now in a repository under your control.

Let's make a simple modification to word count: I would like to know the distribution (counts) of all words that follow the word "perfect". That is, for the phrase "perfect *x*", I want to know how many times each word appears as the *x*, where *x* is any non-zero-length word. To reduce noise, I am not interested in *x*'s that appear only once.

Important: To be clear, use the "words" as generated by the class `io.bespin.java.util.Tokenizer` class in Bespin. This will ensure consistent handling of corner cases involving empty strings, words containing punctuation, etc.

A Simple Modification to Word Count

Create a program called `PerfectX` in the package `ca.uwaterloo.cs451.a0` that implements the specifications above.

```
$ hadoop jar target/assignments-1.0.jar ca.uwaterloo.cs451.a0.PerfectX \
   -input data/Shakespeare.txt -output cs451-bigdatateach-a0-shakespeare
```

You shouldn't need to write more than a couple lines of code (beyond changing class names and other boilerplate). We'll go over the Hadoop API in more detail in class, but the changes should be straightforward.

Answer the following questions:

Question 1. In the Shakespeare collection, what is the most frequent *x* and how many times does it appear? (Answer this question with command-line tools.)

You can run the above instructions using `check_assignment0_public_linux.py` (assignments/check_assignment0_public_linux.py) as follows:

```
$ wget https://www.student.cs.uwaterloo.ca/~cs451/assignments/check_assignment0_public_linux.py
$ chmod +x check_assignment0_public_linux.py
$ ./check_assignment0_public_linux.py bigdatateach
```

We'll be using exactly this script to check your assignment in the Linux Student CS environment (obviously, with your username instead of `bigdatateach`). **Important:** Make sure that your code runs there even if you do development on your own machine.

Using the Datasci Cluster

The software page (software.html) has details on getting started with the Datasci cluster. Make sure you've properly set up the proxy to view the cluster Resource Manager (RM) webapp. Getting access to the RM webapp is important—you'll need it to track your job status and for debugging purposes.

Once you've ssh'ed into the Datasci cluster, you should be able to run your copy of the word count program (copied over from Bespin) from your assignments repo `bigdata2023f/`:

```
$ hadoop jar target/assignments-1.0.jar ca.uwaterloo.cs451.a0.WordCountSimple \
   -input /data/cs451/enwiki-20180901-sentences-0.1sample.txt -output wc-wiki
```

Note that we're running word count over a larger collection here: a 10% sample of English Wikipedia totaling 1.87 GB (here's a chance to exercise your newly-acquired HDFS skills to confirm for yourself).

Question 2. Run word count on the Datasci cluster and make sure you can access the Resource Manager webapp. What is your application id? It looks something like `application_XXXXXXXXXXXXX_XXXX` and can be found in the Resource Manager webapp. If you ran word count multiple times, any id will do.

Question 3. For this word count job, how many mappers ran in parallel?

Question 4. From the word count program, how many times does "waterloo" appear in the sample Wikipedia collection?

Now run your `PerfectX` program on the sample Wikipedia data:

```
$ hadoop jar target/assignments-1.0.jar ca.uwaterloo.cs451.a0.PerfectX \
   -input /data/cs451/enwiki-20180901-sentences-0.1sample.txt -output cs451-bigdatateach-a0-wiki
```

Question 5. In the sample Wikipedia collection, what are the 10 most frequent *x*'s and how many times does each appear? (Answer this question with command-line tools.)

Note that the Datasci cluster is a shared resource, and how fast your jobs complete will depend on how busy it is. You're advised to begin the assignment early as to avoid long job queues. "I wasn't able to complete the assignment because there were too many jobs running on the cluster" will not be accepted as an excuse if your assignment is late.

You can run the above instructions using `check_assignment0_public_datasci.py` (assignments/check_assignment0_public_datasci.py) as follows:

```
$ wget https://www.student.cs.uwaterloo.ca/~cs451/assignments/check_assignment0_public_datasci.py
$ chmod +x check_assignment0_public_datasci.py
$ ./check_assignment0_public_datasci.py bigdatateach
```

We'll be using exactly this script to check your assignment on the Datasci cluster (with your username instead of `bigdatateach`).

Turning in the Assignment

At this point, you should have a Git repo `bigdata2023f/` and inside the repo, you should have the word count program copied over from Bespin, and the new perfect *x* count implementation, along with your `pom.xml`. Commit these files. Next, create a file called `assignment0.md` inside `bigdata2023f/`. In that file, put your answers to the above questions (1—5). Use the Markdown annotation format: here's a simple guide (http://daringfireball.net/projects/markdown/basics).

Note: DO NOT commit `data/` or `target/` (or any results that you may have generated), so your repo should be very compact — it should only have five files: two Java source files, `pom.xml`, and `assignment0.md`. You can add a `.gitignore` file if you wish. Make sure you do not add assignment?_marks.md to .gitignore, because it prevents us from pushing your mark report to the repository

For this and all subsequent assignments, make sure everything is on the default branch of your repo. (This will typically be called "main", but the name doesn't matter as long as it's the default). Push your repo to GitLab. You can verify that it's there by logging into your GitLab account in a web browser: your assignment should be viewable in the web interface.

This and subsequent assignments contain two parts, one that can be completed using the single-node Hadoop cluster (i.e., local model), and another that requires the Datasci cluster. For the first, make sure that your code runs in the Linux Student CS environment (even if you do development on your own machine), which is where we will be doing the marking. "But it runs on my laptop!" will not be accepted as an excuse if we can't get your code to run.

Giving permission to CS451 account

Almost there! Add the user **cs451** to your repository as a "Maintainer". To do this, on the project page, from the left menu select Settings>Members. On the "Project members" page, search for "cs451" under "GitLab member or Email address". Then, under "Choose a role permission" select "Maintainer". Leave the "Access expiration date" empty. Click on "Invite".

And that's it!

To give you an idea of how we'll be marking this and future assignments—we will clone your repo and use the above check scripts:

- `check_assignment0_public_linux.py` (assignments/check_assignment0_public_linux.py) in the Linux Student CS environment.
- `check_assignment0_public_datasci.py` (assignments/check_assignment0_public_datasci.py) on the Datasci cluster.

We'll make sure the data files are in the right place, and once the code completes, we will verify the output. It is highly recommended that you run these check scripts: if it doesn't work for you, it won't work for us either.

Grading

This assignment is worth a total of 20 points, broken down as follows:

- The questions above are worth a total of 10 points.
- Getting your code to compile and successfully run is worth another 8 points (4 points each for `PerfectX` in the Linux student CS environment and on the Datasci cluster). We will make a minimal effort to fix *trivial* issues with your code (e.g., a typo)—and deduct points—but **will not** spend time debugging your code. It is your responsibility to make sure your code runs: we have taken care to specify exactly how we will run your code—if anything is unclear, it is your responsibility to seek clarification. In order to get a perfect score of 8 for this portion of the grade, we should be able to run the two public check scripts above successfully without any errors.
- Another 2 points is allotted to us verifying the output of your program in ways that we will not tell you. We're giving you the "public" versions of the check scripts; we'll run a "private" version to examine your output further (i.e., think blind test cases).

Reference Running Times

To help you gauge the efficiency of your solution, we are giving you the running times of our reference implementations. Keep in mind that these are machine dependent and can vary depending on the server/cluster load.

Class name	Running time Linux	Running time Datasci
WordCountSimple	30 seconds	5 minutes
PerfectX	55 seconds	60 seconds

Back to top