
Assignment 1: MapReduce (7pt)

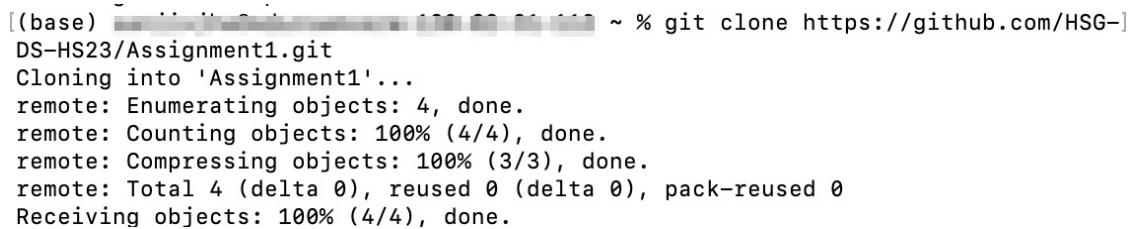
Deadline: Oct 03, 2023; 23:59 CET

In this 2-week assignment, you will learn the basics of distributed computing by implementing MapReduce. You will also gain hands-on experience with gRPC, an open-source and high-performance framework for Remote Procedure Call (RPC), and with Git, a distributed version control system. For this assignment, we provide a Java¹ project template such that you can focus on the learning objectives.

- ① **Setup Git (1pt)** Set up Git on your local device by following this tutorial². Your first task is to clone the Git repository containing the package for this assignment, which is available on GitHub³. To solve this assignment, please fork the Git repository and use the resulting link to perform this task. You can do so by logging into your GitHub account in your web browser, then locating the assignment repository, and forking it into your account. You can then clone the repository by running the following command:

```
git clone link/to/your/forked/repository
```

If successful, this command will create a copy of the assignment package on your machine. To complete this task, send us a screenshot `task1.jpg` showing the successful execution of the `git clone` command (see Figure 1 and the hand-in instructions).



```
[(base) ~] % git clone https://github.com/HSG-DS-HS23/Assignment1.git
DS-HS23/Assignment1.git
Cloning into 'Assignment1'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Figure 1: Successful creation of a local copy of a remote Git repository.

- ② **Computing word counts with MapReduce (2pt)**

MapReduce⁴ handles large computations efficiently by distributing and performing tasks over multiple worker servers simultaneously: It divides the input data into smaller chunks and processes them separately in parallel, making the process faster and scalable over multiple machines. For performing MapReduce, traditionally, a large file gets divided into smaller chunks, and then the map task is performed on multiple server machines. Finally, the reduce function collects all the output from multiple machines to produce a consolidated result.

In this task, you will learn and demonstrate similar behavior for counting words in a given text file on your local machine. Please use the initial code and the input file `pigs.txt` provided to you in the assignment package. You are required to complete the implementation of the `map` and `reduce` functions from the project template:

¹Tutorial for installing Java <https://www.geeksforgeeks.org/download-and-install-java-development-kit-jdk-on-windows-mac-and-linux/>, <https://www.oracle.com/java/technologies/downloads/>

²<https://docs.github.com/en/get-started/quickstart/set-up-git>

³<https://github.com/HSG-DS-HS23/Assignment1.git>

⁴<http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf>

I the `map` function assigns each word with an initial frequency value as 1 (for example, key-value pair `The:1`) and saves them into a text file.

II the `reduce` function consolidates all of the output files produced from the map function and assigns the final word count key-value pairs, for example, `The:64`. Finally, it sorts them into descending order, with the most frequent word first. An example expected output of the method can be found in the file `output.txt` and you can compare your output using the `CheckOutput.java` file available in the assignment package.

To run the main method in `MapReduce.java`, see `README.md` and follow the commands. Once you have successfully updated, compiled, and run the Java project, please submit your solution code (i.e., the updated project template) with the output file as `output-task2.txt` in the submission package.

③ An RPC-based implementation of MapReduce (3pt)

So far, you have been running your MapReduce implementation as a single process on your machine. In this task, you will use a Remote Procedure Call to distribute the implementation across multiple processes. For practical reasons, in this task, you will continue to run the project on a single machine. Still, at this point, you could easily distribute the computation across multiple machines. We will return to distributing computation across multiple machines in Week 5.

Create a simple gRPC-based implementation for performing MapReduce on two worker servers (that could be remote).⁵ To simulate a distributed system on your local machine, you will create two worker servers and a client node. You can reuse the MapReduce implementation and the input file `pigs.txt` from *Task 2* — and the result of the task should be the same as *Task 2*.

To complete this task and implement a client-server architecture using gRPC, update the `MrClient.java`, `MrMapServer.java`, and `MrReduceServer.java` files. The worker servers accept map and reduce requests from the client channel stubs to perform MapReduce on the server side. To communicate between the client and the servers, `communicate.proto` defines the messages and the type of service calls. Here, the client streams the file chunks to the map worker and keeps track of the job status. Whenever a file is successfully mapped, the map worker sends the value, for example, 2 to the client, indicating task completion. After performing all mapping tasks, the client node makes a unary call for the reduce function, and the reduce worker performs it. Once completed, it sends the job status back to the client. After waiting a few more seconds for new tasks, the client disconnects. However, the server keeps on listening. To implement gRPC through the given code snippet, perform the following setup steps from `README.md`. For submission, submit your updated code with the output file `output-task3.txt` in the submission package.

④ Basics of RPC and MapReduce (1pt)

Please submit your answers to the following questions in the `Report.md` file.

- 1.) (0.25pt) What are Interface Definition Languages (IDL) used for? Name and explain the IDL you use for Task 3.
- 2.) (0.25pt) In this implementation of gRPC, you use *channels*. What are they used for?
- 3.) (0.5pt) Describe how the MapReduce algorithm works. Do you agree that the MapReduce programming model may have latency issues? What could be the cause of this? Can this programming model be suitable (recommended) for iterative machine learning or data analysis applications? Please explain your argument.

⁵To get familiar with gRPC: <https://grpc.io/docs/languages/java/basics/>

Hand-in Instructions By the deadline, you should hand in a single **zip** file via Canvas upload. The name of this file should start with **a1** and contain the last names of all team members separated by underscores (e.g., **a1_jha_lemee_ciortea.zip**). It should **only** contain the following files:

- Screenshot for task one **task1.jpg**
- Solution code package (if you want to submit your code as a zip file) and output files **output-task2.txt**, **output-task3.txt**
- All answers to the assignment questions of **Task 4** and the GitHub link your (forked) repository in the given **REPORT.md**

Across all tasks in this and the other assignments in this course, you are **required to declare** any support that you received from others and, within reasonable bounds,⁶ any support tools that you were using while solving the assignment.

⁶It is not required that you declare that you were using a text-editing software with orthographic correction; it is however required to declare if you were using any non-standard tools such as generative machine learning models such as GPT