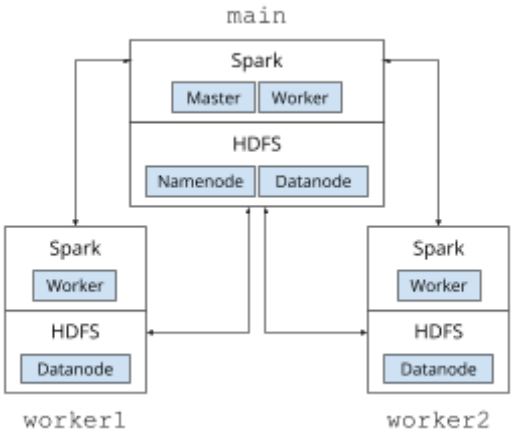
CS 511 Fall 2025: Project 1 HDFS/Spark

Due date: Oct 6 2025 11:59:59 PM Central Time

You have just started as an engineer at Tera Sorting Cap, a company specializing in sorting digital caps by their serial numbers. The company is struggling to *scale up* its machines to keep up with the ever-growing dataset. "We can't sort the dataset fast enough," panicked your fellow engineer, "What's worse is that our dataset is about to outgrow the largest hard disk in the world."

"What about *scaling out*?" you said. Thanks to your recent reads on HDFS and Spark, you are confident that you can store and sort the dataset across many commodity machines.

Upon extended discussions, your company has tasked you to produce a proof of concept: **deploy a Docker cluster with 1 main container and 2 worker containers having HDFS as the storage and Spark as the analytic engine**.



Your support team has drafted the steps and provided scripts to test along the way.

Deliverable and Grading

https://github.com/yongjoo-classroom/cs511-fall2025-p1 contains the solution template and grading scripts. Please copy the solution template and make necessary changes. If

you're going to fork it, remember to set the new repository as private. **Finally, please submit a zip file for your code, name it as cs511-fall2025-p1-netid.**

The solution template may be updated over time (it should be minor updates). Please keep an eye out for it.

The zip file must be submitted on Canvas by the due date noted at the beginning of this document. Late submission and/or any update to the repository after the deadline gets a 20% point reduction of the earned grade for each late day. Submissions will not get grades if more than 4 days late. If any special accommodation is needed, please contact CS 511 staff with supporting documents at least 3 days before the due date.

We are working on assigning everyone a VM and will let you know once it is ready. For most tasks, however, working locally on your laptop should be more than sufficient to complete this project—and in many cases, even easier.

Only these files can be modified:

1. Dockerfiles: `cs511p1-common.Dockerfile`, `cs511p1-main.Dockerfile`, and `cs511p1-worker.Dockerfile`
2. Setup scripts: `setup-main.sh` and `setup-worker.sh`
3. Startup scripts: `start-main.sh` and `start-worker.sh`
4. Other files **only** if stated below
5. New files (if needed)

Our grader will execute the following commands on Ubuntu 22.04 Bash in order.

```
cd cs511-fall2025-p1-netid
bash start-all.sh
bash test-all.sh
```

Grading rubric for a score out of 120, scaled to 100%, plus additional 20 extra credits:

Criteria	Score
Start up successfully	20
Pass HDFS tests (test_1_hdfs.sh)	40
Pass Spark tests (test_2_spark.sh)	40
Write a Spark demo for Tera Sorting Cap	20

Write a Spark demo for PageRank	20 (extra credit)

Setup

Install Docker https://www.docker.com/get-started/ if not yet. You can test your installation by running the following command on your command prompt which should print a message starting with "Hello from Docker!".

```
docker run hello-world
```

Part 0: Introduction to Docker

Apart from grading scripts, the solution template has set up a multi-node deployment with network capability and exchanged SSH keys between nodes. To get started, execute this command **in one terminal**, called terminal-A:

```
bash start-all.sh
```

Internally, this script builds the Docker images based on their `*.Dockerfile` specifications: `cs511p1-common`, `cs511p1-main`, and `cs511p1-worker`. Lastly, the script spins up a cluster of 3 nodes (`main`, `worker1`, `worker2`) based on `cs511p1-compose.yaml` setup.

After the cluster starts successfully, in **a new terminal**, called terminal-B, you can issue a command to a node via:

```
docker compose -f cs511p1-compose.yaml exec <node> <command> [args ...]
```

For example, to access the `main` Bash command prompt.

```
docker compose -f cs511p1-compose.yaml exec main bash
```

Expected output is similar to:

```
root@main:/#
```

Thanks to Docker's DNS services, these nodes can address other nodes using their hostnames specified in `cs511p1-compose.yaml`.

```
docker compose -f cs511p1-compose.yaml exec worker1 ssh worker2 -t "hostname"
```

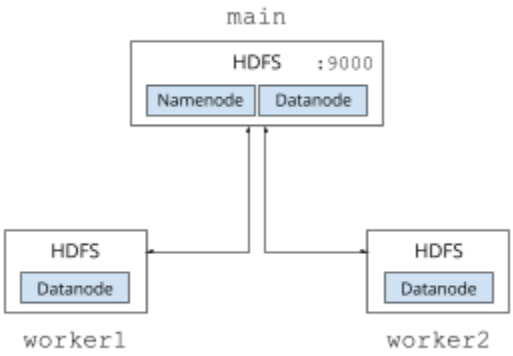
Expected output is similar to:

```
The authenticity of host 'worker2 (xxx.xxx.xxx.xxx)' can't be established.
ECDSA key fingerprint is SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'worker2,xxx.xxx.xxx.xxx' (ECDSA) to the list of known
hosts.
worker2
Connection to worker2 closed
```

To spin down the cluster, interrupt the process that runs `start_all.sh` (e.g., Ctrl-C in terminal-A).

Part 1: HDFS, Scaling Out Storage

After this part, your solution should deploy HDFS across 3 nodes like below. The Hadoop version must be `hadoop-3.3.6`. HDFS must be exposed at `hdfs://main:9000`. Commands `hdfs` and `hadoop` should be accessible in the `main` Bash.



1. Edit `cs511p1-common.Dockerfile`, `cs511p1-main.Dockerfile`, and `cs511p1-worker.Dockerfile` to set up HDFS resources with Docker cache (e.g., installing dependencies, downloading packages, extracting packages).
2. Edit `setup-main.sh` and `setup-worker.sh` to install HDFS resources and configurations on main and workers before their startup.
3. Edit `start-main.sh` and `start-worker.sh` to initialize HDFS services at startup.
4. Add new files if needed.

The deployment should be able to pass these 4 tests.

Q1. (10 points) HDFS should report 3 live datanodes on `main`, `worker1`, and `worker2` with normal status.

```
docker compose -f cs511p1-compose.yaml exec main hdfs dfsadmin -report
```

Q2. (10 points) HDFS should write and read a file correctly.

```
docker compose -f cs511p1-compose.yaml cp resources/fox.txt main:/test_fox.txt
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    hdfs dfs -mkdir -p /test; \
    hdfs dfs -put -f /test_fox.txt /test/fox.txt; \
    hdfs dfs -cat /test/fox.txt'
```

Q3. (10 points) HDFS should execute the namenode benchmark successfully.

```
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    hadoop org.apache.hadoop.hdfs.server.namenode.NNThroughputBenchmark -fs
hdfs://main:9000 -op create -threads 100 -files 10000; \
```

```
    hadoop org.apache.hadoop.hdfs.server.namenode.NNThroughputBenchmark -fs
hdfs://main:9000 -op open -threads 100 -files 10000; \
    hadoop org.apache.hadoop.hdfs.server.namenode.NNThroughputBenchmark -fs
hdfs://main:9000 -op delete -threads 100 -files 10000; \
    hadoop org.apache.hadoop.hdfs.server.namenode.NNThroughputBenchmark -fs
hdfs://main:9000 -op rename -threads 100 -files 10000'
```

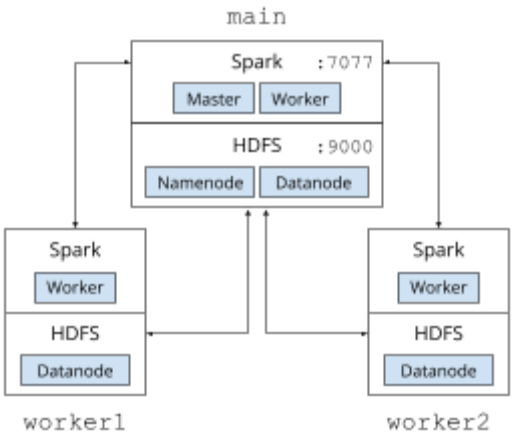
Q4. (10 points) HDFS should execute the TeraSort benchmark successfully. This should take ~200MB of disk space.

```
docker compose -f cs511p1-compose.yaml cp resources/hadoop-terasort-3.3.6.jar \
    main:/hadoop-terasort-3.3.6.jar
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    hdfs dfs -rm -r -f tera-in tera-out tera-val; \
    hadoop jar /hadoop-terasort-3.3.6.jar teragen 1000000 tera-in; \
    hadoop jar /hadoop-terasort-3.3.6.jar terasort tera-in tera-out; \
    hadoop jar /hadoop-terasort-3.3.6.jar teravalidate tera-out tera-val; \
    hdfs dfs -cat tera-val/*;'
```

Equivalently, `bash test_hdfs.sh` runs all these tests and checks the result. Outputs and errors from each query can be inspected under the `out/` directory.

Part 2: Spark, Speeding Up Engine

Finally, your solution should deploy Spark across 3 nodes on top of a functioning HDFS cluster. The spark version must be `spark-3.4.1`. The Spark master must be exposed at `spark://main:7077`. Commands `spark-submit` and `spark-shell` should be accessible in the `main` Bash.



1. Edit `cs511p1-common.Dockerfile`, `cs511p1-main.Dockerfile`, and `cs511p1-worker.Dockerfile` to set up Spark resources with Docker cache (e.g., installing dependencies, downloading packages, extracting packages).
2. Edit `setup-main.sh` and `setup-worker.sh` to install Spark resources and configurations on main and workers before their startup.
3. Edit `start-main.sh` and `start-worker.sh` to initialize Spark services at startup.
4. Add new files if needed.

Q1. (10 points) Spark context should contain 3 executors on `main`, `worker1`, and `worker2`.

```
docker compose -f cs511p1-compose.yaml cp resources/active_executors.scala \
    main:/active_executors.scala
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    cat /active_executors.scala | spark-shell --master spark://main:7077'
```

Q2. (10 points) Spark should execute the Pi estimation benchmark successfully.

```
docker compose -f cs511p1-compose.yaml cp resources/pi.scala main:/pi.scala
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    cat /pi.scala | spark-shell --master spark://main:7077'
```

Q3. (10 points) Spark should read from HDFS correctly.

```
docker compose -f cs511p1-compose.yaml cp resources/fox.txt main:/test_fox.txt
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    hdfs dfs -mkdir -p /test; \
```

```
    hdfs dfs -put -f /test_fox.txt /test/fox.txt; \
    hdfs dfs -cat /test/fox.txt'
docker compose -f cs511p1-compose.yaml exec main bash -x -c '\
    echo "sc.textFile(\"hdfs://main:9000/test/fox.txt\").collect()" | \
    spark-shell --master spark://main:7077'
```

Q4. (10 points) Spark should execute the TeraSort benchmark successfully. This should take ~200MB of disk space.

```
docker compose -f cs511p1-compose.yaml cp resources/spark-terasort-1.2.jar \
    main:/spark-terasort-1.2.jar
docker compose -f cs511p1-compose.yaml exec main spark-submit \
    --master spark://main:7077 \
    --class com.github.ehiggs.spark.terasort.TeraGen /spark-terasort-1.2.jar \
    100m hdfs://main:9000/spark/tera-in
docker compose -f cs511p1-compose.yaml exec main spark-submit \
    --master spark://main:7077 \
    --class com.github.ehiggs.spark.terasort.TeraSort /spark-terasort-1.2.jar \
    hdfs://main:9000/spark/tera-in hdfs://main:9000/spark/tera-out
docker compose -f cs511p1-compose.yaml exec main spark-submit \
    --master spark://main:7077 \
    --class com.github.ehiggs.spark.terasort.TeraValidate /spark-terasort-1.2.jar \
    hdfs://main:9000/spark/tera-out hdfs://main:9000/spark/tera-val
```

Equivalently, `bash test_spark.sh` runs all these tests and checks the result. Outputs and errors from each query can be inspected under the `out/` directory.

Part 3: HDFS/Spark Sorting

Conveniently, Tera Sorting Cap locally stores its dataset of caps in a large CSV file where each line represents properties of a cap (year of manufacture and serial number). An example of the CSV file:

```
1999,1234-5678-91011
1800,1001-1002-10003
2023,0829-0914-00120
2050,9999-9999-99999
```

Complete your proof-of-concept HDFS/Spark cluster with a demo sorting the above example dataset. Please see the sorting instructions below.

(20 points) Import the file from a local file system to HDFS. Then, in a Spark application (e.g., a Scala program), remove caps made in the future (year > 2025) and sort all the caps from newest to oldest, then from lowest to highest serial number. The grader script is partially provided with the ground truth. You are required to complete it to validate your sorting.

Part 4: HDFS/Spark PageRank (Extra Credit)

As a bonus, there is a task of implementing the PageRank algorithm that measures the importance of nodes in large networks. Please refer to the wiki page of the algorithm https://en.wikipedia.org/wiki/PageRank. In this task, the simplified algorithm with a damping factor should be implemented.
Each node of the network is uniquely identified as an integer. A network dataset is a CSV file where each line represents that node i has an outgoing link to node j. An example of the CSV file:

```
2,3
3,2
4,2
5,2
5,6
6,5
7,5
8,5
9,5
10,5
11,5
4,1
```

Complete your proof-of-concept HDFS/Spark cluster with a demo computing this dataset.

(20 points) Import the file from a local file system to HDFS. Then, in a Spark application (e.g., a Scala program), compute the PageRank value for each node using a damping factor *d*=0.85 and a tolerance of ε=0.0001. Output results in a CSV file. Each line contains the node and its PageRank value. Sort lines in descending order of PageRank values (keep 3 decimal places). If there is a tie, then please break the tie in ascending order of node integer values. The grader script is partially provided with the ground truth of PageRank value. You are required to complete it to validate your PageRank.

FAQs

Should we use laptops or are clusters provided?

We have requested VMs for Engineering IT so each student can be assigned a VM. For most tasks, however, working locally (on a laptop) would be sufficient to complete this project, and also, even easier.

Is there a way to see Spark's Web UI to check a cluster status and job progress?

Yes, once Spark starts successfully, visit `localhost:8080` and `localhost:8081` on your browser to access the master and worker's Web UIs respectively. These ports are exposed by Docker as specified in cs511p1-compose.yaml

Docker Compose starts with 2 workers but we need to set up 3 HDFS datanodes and 3 Spark workers. Where should the third worker be?

The third worker should be on the `main` container (see diagrams).

If the bug still occurs in the latest version, please report the issue on Canvas. We will continue to patch the test scripts as we discover more cases. The list of patches will be announced on a Campuswire note.

How do you fix out-of-memory errors?

The *default* configuration for HDFS/Spark has a minimum memory requirement. We suggest attempting these in order: 1) fulfill Docker's memory requirement, 2) reconfigure the cluster memory usage, or 3) move to a machine with a larger memory.

docker-compose or docker compose?

These two commands are simply two versions of the same thing. The former, docker-compose is the compose command version 1, and is no longer included Docker Desktop. The latter is version 2 and is preferred. We are planning to switch to version 2, docker compose. If docker-compose brings some dependency issues, you can switch to docker compose.

Programming language requirement?

There is no specific programming language requirement. You can choose your preferred language to complete this assignment, e.g., using PySpark, or Scala.