Effat University – College of Engineering – Department of Computer Science

CS4074: Big Data Analytics
Lab 2: Exploring Hadoop & HDFS

Student Name:	Sana Rahmani	**Student ID:**	S21107268
Section:	1	**Date:**	15/2/2026

Lab Overview

Objective:	Gain hands-on experience with Hadoop HDFS: file operations, block management, replication, fault tolerance, and storage design.
Prerequisites:	Hadoop installed and configured (Lab 1).
Duration:	2 hours (Parts 1–5: guided | Part 6: deep work | Part 7: deliverables).
Tools:	Terminal, Hadoop 3.x, Web browser (HDFS Web UI).

Lab Structure: Part 1: Start Services (10 min) → Part 2: HDFS Commands (20 min) → Part 3: Blocks & Replication (20 min) → Part 4: Admin & Health (15 min) → Part 5: Saudi Open Data Exercise (25 min) → Part 6: Deep Work (30 min) → Part 7: Deliverables (10 min)

Part 1: Verify Hadoop Installation & Start Services (10 min)

Before working with HDFS, we need to ensure Hadoop is properly running.

1.1 Step 1: Check Versions

Check Java & Hadoop

```
# Verify Java
java -version

# Verify Hadoop
hadoop version
```

Task 1.1

Run both commands and record the versions below:

- Java version: 1.8.0_472
- Hadoop version: 3.3.6

1.2 Step 2: Start HDFS Services

Start HDFS

```
# Format NameNode (ONLY if first time -- do NOT repeat!)
# hdfs namenode -format

# Start HDFS daemons
start-dfs.sh

# Verify running processes
jps
```

You should see at least: `NameNode`, `DataNode`, and `SecondaryNameNode`.

1.3 Step 3: Access the Web UI

Open your browser and navigate to: `http://localhost:9870`

Task 1.2

Take a screenshot of the HDFS Web UI overview page showing the cluster summary.

Common Issue

If `jps` does not show NameNode, try:

```
stop-dfs.sh
rm -rf /tmp/hadoop-*     # Clear temp data
hdfs namenode -format    # Re-format
start-dfs.sh
```

Part 2: Basic HDFS Commands (20 min)

HDFS commands follow the pattern: `hdfs dfs -<command> <arguments>`

2.1 Create a Working Directory

Directory Operations

```
# Create your lab directory in HDFS
hdfs dfs -mkdir /user
hdfs dfs -mkdir /user/student
hdfs dfs -mkdir -p /user/student/lab2/input
```

```
# List directories
hdfs dfs -ls /user/student/
hdfs dfs -ls -R /user/student/
```

2.2 Upload Files to HDFS

First, create a sample text file on your local machine:

Create Sample File

```
# Create a sample file locally
echo "Big Data Analytics is the future of computing.
Hadoop provides distributed storage with HDFS.
Apache Spark processes data 100x faster than MapReduce.
Saudi Vision 2030 embraces digital transformation.
Effat University leads in technology education." > ~/sample.txt

# Upload to HDFS
hdfs dfs -put ~/sample.txt /user/student/lab2/input/

# Alternative command (same result)
hdfs dfs -copyFromLocal ~/sample.txt /user/student/lab2/input/sample2.
    txt
```

2.3 Read & Browse Files

Reading Files in HDFS

```
# Display full file content
hdfs dfs -cat /user/student/lab2/input/sample.txt

# Show first 1KB
hdfs dfs -head /user/student/lab2/input/sample.txt

# Show last 1KB
hdfs dfs -tail /user/student/lab2/input/sample.txt

# Count lines, words, bytes
hdfs dfs -cat /user/student/lab2/input/sample.txt | wc -l
```

2.4 Download & Delete Files

Download & Cleanup

```
# Download from HDFS to local
hdfs dfs -get /user/student/lab2/input/sample.txt ~/downloaded.txt

# Copy within HDFS
hdfs dfs -cp /user/student/lab2/input/sample.txt /user/student/lab2/
    backup.txt

# Move / Rename
```

```
hdfs dfs -mv /user/student/lab2/backup.txt /user/student/lab2/renamed.
    txt

# Delete a file
hdfs dfs -rm /user/student/lab2/renamed.txt

# Delete a directory recursively
# hdfs dfs -rm -r /user/student/lab2/old_folder
```

Task 2.1

Complete the following exercises:

a) Create a directory: `/user/student/lab2/output`
b) Upload `sample.txt` into it
c) Display the contents using `-cat`
d) Count the number of words in the file
e) Download the file back to your local machine as `result.txt`

Write the exact commands you used:

```
hdfs dfs -put ~/sample.txt /user/student/lab2/input/
hdfs dfs -cat /user/student/lab2/input/sample.txt
hdfs dfs -cat /user/student/lab2/input/sample.txt | wc -w
hdfs dfs -get /user/student/lab2/input/sample.txt ~/downloaded.txt
```

HDFS Command Reference

`-mkdir -p`	Create directory (with parents)
`-put / -copyFromLocal`	Upload local → HDFS
`-get / -copyToLocal`	Download HDFS → local
`-cat / -head / -tail`	Read file contents
`-ls / -ls -R`	List files (recursive)
`-cp / -mv`	Copy / Move within HDFS
`-rm / -rm -r`	Delete file / directory
`-du -h / -df -h`	Disk usage / free space
`-stat %r %b`	Show replication factor / block size
`-chmod / -chown`	Change permissions / ownership

Part 3: Understanding Blocks & Replication (20 min)

In Lecture 2 you learned that HDFS splits files into **128 MB blocks** and replicates each block **3 times**. Now let's see this in action.

3.1 Step 1: Generate a Large File

Generate a 300 MB File

```
# Generate a 300 MB file filled with random text
dd if=/dev/urandom of=~/bigfile.dat bs=1M count=300

# Verify size
```

```
ls -lh ~/bigfile.dat

# Upload to HDFS
hdfs dfs -put ~/bigfile.dat /user/student/lab2/input/
```

3.2 Step 2: Inspect Blocks

Check Block Distribution

```
# See how HDFS split the file into blocks
hdfs fsck /user/student/lab2/input/bigfile.dat -files -blocks

# Check replication factor
hdfs dfs -stat %r /user/student/lab2/input/bigfile.dat

# Check block size
hdfs dfs -stat %b /user/student/lab2/input/bigfile.dat
```

Task 3.1

Record the output from `hdfs fsck`:

- Total file size: bytes 314.572.800
- Number of blocks: 3
- Replication factor: 1
- Total replicated blocks: 3

Reflection 3.1

A 300 MB file with 128 MB block size should produce how many blocks? Show your calculation. Does this match your `fsck` output?

File size = 300 MB
Block size = 128 MB

300 ÷ 128 = 2.34

Since HDFS cannot create partial blocks smaller than the remaining data, we round up to the next whole number.

Number of blocks = 3 blocks

3.3 Step 3: Change Replication Factor

Modify Replication

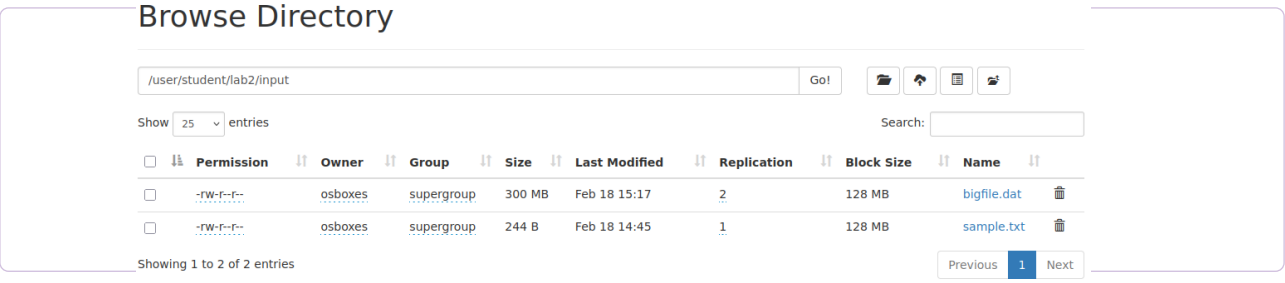
```
# Change replication factor to 2
hdfs dfs -setrep 2 /user/student/lab2/input/bigfile.dat

# Verify the change
hdfs dfs -stat %r /user/student/lab2/input/bigfile.dat

# Check under-replicated blocks (if any)
hdfs fsck /user/student/lab2/input/bigfile.dat -files -blocks
```

Task 3.2

a) What was the replication factor before? 1

b) What is it now? 2

c) Open the Web UI (`http://localhost:9870`) → Utilities → Browse the file system. Take a screenshot showing the file and its replication.



Part 4: HDFS Admin & Health Check (15 min)

As a system administrator, you need to monitor HDFS health.

4.1 Cluster Report

HDFS Admin Report

```
# Full cluster report
hdfs dfsadmin -report
```

Task 4.1

From the `dfsadmin -report` output, fill in:

- Total configured capacity: 216.08 GB
- DFS used: 302.42 MB
- DFS remaining: 193.33 GB
- Number of live DataNodes: 1

4.2 Disk Usage

Disk Usage Commands

```
# Show disk usage of your files (human readable)
hdfs dfs -du -h /user/student/lab2/

# Show total HDFS capacity
hdfs dfs -df -h
```

4.3 Safe Mode

Safe Mode Operations

```
# Check if in safe mode
hdfs dfsadmin -safemode get

# Enter safe mode (read-only -- no writes allowed)
hdfs dfsadmin -safemode enter
```

```
# Try uploading -- this should FAIL!
echo "test" > /tmp/test.txt
hdfs dfs -put /tmp/test.txt /user/student/lab2/

# Leave safe mode
hdfs dfsadmin -safemode leave

# Now the upload should work
hdfs dfs -put /tmp/test.txt /user/student/lab2/
```

Task 4.2

a) Enter safe mode and attempt to upload a file. What error message do you get?

Cannot create file /user/student/lab2/test.txt._COPYING_. Name node is in safe mode.

b) Why would an administrator use safe mode? Give one real-world scenario.

An administrator uses safe mode to make HDFS read-only while checking system stability.

Example: After a power failure, safe mode is used until the NameNode verifies all blocks are healthy.

4.4 File System Check

HDFS Health Check

```
# Check entire HDFS for corrupt/missing blocks
hdfs fsck / -files -blocks

# Summary only
hdfs fsck /
```

Reflection 4.1

What does `hdfs fsck` report about the health status? Are there any corrupt or missing blocks?

The hdfs fsck / command reported that the filesystem is HEALTHY.
There are no corrupt blocks and no missing blocks,
which means the HDFS cluster is functioning correctly and all data blocks are properly replicated and accessible.

Part 5: Real-World Exercise: Saudi Open Data (25 min)

In this part, you will work with real data and organize it in HDFS like a professional data engineer.

5.1 Step 1: Prepare the Data

Generate a Realistic Dataset

```
# Create a Saudi cities population dataset (CSV)
cat > ~/saudi_population.csv << 'EOF'
city,region,population_2024,area_km2,density
Riyadh,Riyadh,7500000,1798,4171
Jeddah,Makkah,4700000,1686,2788
Makkah,Makkah,2200000,760,2895
```

```
Madinah,Madinah,1500000,589,2547
Dammam,Eastern,1300000,800,1625
Khobar,Eastern,650000,750,867
Tabuk,Tabuk,600000,780,769
Abha,Asir,500000,370,1351
Taif,Makkah,700000,321,2181
Buraidah,Qassim,600000,980,612
EOF

# Verify
cat ~/saudi_population.csv
wc -l ~/saudi_population.csv
```

5.2 Step 2: Create Organized HDFS Structure

Organize Data in HDFS

```
# Create a professional directory structure
hdfs dfs -mkdir -p /data/saudi/population
hdfs dfs -mkdir -p /data/saudi/weather
hdfs dfs -mkdir -p /data/saudi/healthcare

# Upload the dataset
hdfs dfs -put ~/saudi_population.csv /data/saudi/population/

# Verify the structure
hdfs dfs -ls -R /data/saudi/

# View file in HDFS
hdfs dfs -cat /data/saudi/population/saudi_population.csv
```

5.3 Step 3: Analyze with Basic Commands

Basic Data Analysis via CLI

```
# Count total records (minus header)
hdfs dfs -cat /data/saudi/population/saudi_population.csv | tail -n +2 |
    wc -l

# Find cities in Makkah region
hdfs dfs -cat /data/saudi/population/saudi_population.csv | grep "Makkah
   "

# Sort by population (descending)
hdfs dfs -cat /data/saudi/population/saudi_population.csv | tail -n +2 |
    \
    sort -t',' -k3 -nr

# Show only city names and populations
hdfs dfs -cat /data/saudi/population/saudi_population.csv | tail -n +2 |
    \
    cut -d',' -f1,3
```

Task 5.1

Using HDFS commands, answer the following:

a) How many cities are in the dataset? 10

b) Which city has the highest population? Riyadh

c) How many cities are in the Makkah region? 3

d) What is the block count for this CSV file? (`hdfs fsck`) 1

e) Why is the CSV stored in only 1 block? Because the CSV is very small (only a few hundred bytes) and the default HDFS block size is 128 MB, so the entire file fits inside a single block.

Task 5.2

Take a screenshot of the HDFS Web UI showing the `/data/saudi/` directory tree.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	osboxes	supergroup	0 B	Feb 18 18:05	0	0 B	healthcare
drwxr-xr-x	osboxes	supergroup	0 B	Feb 18 18:31	0	0 B	population
drwxr-xr-x	osboxes	supergroup	0 B	Feb 18 18:05	0	0 B	weather

Part 6: Deep Work – Fault Tolerance & Storage Experimentation (30 min)

About Deep Work

This section challenges you to **investigate HDFS behavior beyond the basics**. You will design experiments, observe system behavior, and draw conclusions that connect to Lecture 2 concepts. Deep work questions require **critical thinking** — there are no single right answers, but your reasoning must be supported by experimental evidence.

6.1 Experiment 6.1: The Small Files Problem

In Lecture 2, we discussed that HDFS struggles with many small files. Let's **prove it**.

Small Files Experiment

```
# Step 1: Record baseline NameNode status
hdfs dfsadmin -report | grep -E "files|blocks|Total"

# Step 2: Generate 1,000 small files (1 KB each)
mkdir -p ~/small_files
for i in $(seq 1 1000); do
    dd if=/dev/urandom of=~/small_files/file_$i.txt bs=1K count=1 2>/dev
       /null
done

# Verify: total size is ~1 MB
du -sh ~/small_files/

# Step 3: Upload all small files to HDFS
hdfs dfs -mkdir -p /user/student/lab2/small_files
hdfs dfs -put ~/small_files/* /user/student/lab2/small_files/
```

```
# Step 4: Check NameNode status again
hdfs dfsadmin -report | grep -E "files|blocks|Total"

# Step 5: Check total blocks used
hdfs fsck /user/student/lab2/small_files/ | grep -E "Total blocks|Total
    size"
```

Now let's compare with an archive approach:

Archive the Small Files

```
# Step 6: Create a Hadoop Archive (HAR)
hadoop archive -archiveName small_files.har -p /user/student/lab2/
    small_files \
     /user/student/lab2/

# Step 7: List the archive contents
hdfs dfs -ls har:///user/student/lab2/small_files.har/

# Step 8: Check blocks used by the archive
hdfs fsck /user/student/lab2/small_files.har/ -files -blocks | \
     grep -E "Total blocks|Total size"
```

Task 6.1 – Record Your Findings

Metric	1,000 Small Files	HAR Archive
Total data size	1,024,000 B (1 MB)	1,024,000 B (1 MB)
Number of HDFS blocks	1000	1000
NameNode entries (files + blocks)	2000	2000

Reflection 6.1

Why does 1 MB of data stored as 1,000 files consume significantly more NameNode resources than a single 1 GB file? Explain using the concept that each file/block entry takes ∼150 bytes of NameNode memory.

Storing 1 MB as 1,000 small files consumes significantly more NameNode memory because each file and block requires metadata (about 150 bytes).
1,000 files create 1,000 file entries and 1,000 block entries,
consuming much more memory than storing the same data as a single archived file with only a few blocks

6.2 Experiment 6.2: Simulating Node Failure

Let's test HDFS fault tolerance by manually corrupting a block.

Fault Tolerance Experiment

```
# Step 1: Upload a test file
echo "This is a critical healthcare record." > ~/critical.txt
hdfs dfs -put ~/critical.txt /user/student/lab2/critical.txt
```

```
# Step 2: Find block locations
hdfs fsck /user/student/lab2/critical.txt -files -blocks -locations

# Step 3: Find the actual block file on the local filesystem
# (Look for the block ID from Step 2)
find /tmp/hadoop-*/dfs/data -name "blk_*" -type f 2>/dev/null | head -10

# Step 4: Corrupt a block by overwriting it
# COPY the block ID from Step 2, then:
# find /tmp/hadoop-*/dfs/data -name "blk_<BLOCK_ID>*" -type f
# echo "CORRUPTED" > <path_to_block_file>

# Step 5: Wait ~30 seconds, then check
sleep 30
hdfs fsck /user/student/lab2/critical.txt -files -blocks

# Step 6: Can you still read the file?
hdfs dfs -cat /user/student/lab2/critical.txt
```

Task 6.2 – Record Your Observations

a) How many replicas did the file have before corruption? 1

b) After corrupting one block, could you still read the file? no

c) Did HDFS detect the corruption? How? Yes. HDFS detects corruption using block reports and checksum verification.

d) Did HDFS re-replicate the block to restore the replication factor? no

Reflection 6.2

How long did recovery take? What would happen if **2 out of 3** replicas were lost simultaneously? At what point does HDFS lose data permanently?

When the replication factor was set to 2 with only one DataNode available,
HDFS reported under-replicated blocks because it could not create the second replica.
After changing the replication factor back to 1, the system became fully healthy with no under-replicated blocks. This demonstrates that HDFS replication depends on the number of available DataNodes.

6.3 Experiment 6.3: Block Size Experiment

Varying Block Size

```
# Upload the SAME 300 MB file with different block sizes

# Block size = 64 MB
hdfs dfs -D dfs.blocksize=67108864 -put ~/bigfile.dat \
    /user/student/lab2/bigfile_64m.dat

# Block size = 128 MB (default)
hdfs dfs -D dfs.blocksize=134217728 -put ~/bigfile.dat \
    /user/student/lab2/bigfile_128m.dat

# Block size = 256 MB
hdfs dfs -D dfs.blocksize=268435456 -put ~/bigfile.dat \
    /user/student/lab2/bigfile_256m.dat

# Compare block counts
```

```
echo "=== 64 MB blocks ==="
hdfs fsck /user/student/lab2/bigfile_64m.dat 2>/dev/null | grep "Total
    blocks"

echo "=== 128 MB blocks ==="
hdfs fsck /user/student/lab2/bigfile_128m.dat 2>/dev/null | grep "Total
    blocks"

echo "=== 256 MB blocks ==="
hdfs fsck /user/student/lab2/bigfile_256m.dat 2>/dev/null | grep "Total
    blocks"
```

Task 6.3 – Fill the Table

File Size	Block Size	Number of Blocks	NameNode Entries	
300 MB	64 MB	5	6	
300 MB	128 MB	3	4	
300 MB	256 MB	2	3	

Reflection 6.3

When would you choose a **smaller** block size vs. a **larger** one? Discuss the trade-offs in terms of: (1) NameNode memory, (2) parallelism in MapReduce, and (3) network overhead.

Smaller block sizes (e.g., 64 MB) increase parallelism in MapReduce because more blocks allow more tasks to run simultaneously. However, they increase NameNode memory usage due to more metadata entries and may increase network overhead.

Larger block sizes (e.g., 256 MB) reduce NameNode metadata and network overhead but decrease parallelism since fewer blocks are available for distributed processing.

Therefore, smaller blocks are useful when high parallel processing is required, while larger blocks are better for reducing metadata overhead and improving sequential read performance.

6.4 Experiment 6.4: Build a Data Lake Directory Structure

Design and implement an HDFS directory structure for a hospital system.

Data Lake Design

```
# Create the hospital data lake structure
hdfs dfs -mkdir -p /datalake/raw/patients
hdfs dfs -mkdir -p /datalake/raw/lab_results
hdfs dfs -mkdir -p /datalake/raw/imaging
hdfs dfs -mkdir -p /datalake/processed
hdfs dfs -mkdir -p /datalake/archive

# Set different replication factors
hdfs dfs -setrep 3 /datalake/raw/imaging
hdfs dfs -setrep 2 /datalake/processed
hdfs dfs -setrep 1 /datalake/archive

# Verify replication settings
echo "--- Imaging replication ---"
hdfs dfs -stat %r /datalake/raw/imaging
echo "--- Processed replication ---"
hdfs dfs -stat %r /datalake/processed
```

```
echo "--- Archive replication ---"
hdfs dfs -stat %r /datalake/archive

# View the full tree
hdfs dfs -ls -R /datalake/
```

Task 6.4

a) Take a screenshot of `hdfs dfs -ls -R /datalake/` output.
b) Upload a sample file to each of: `/datalake/raw/patients/`, `/datalake/raw/imaging/`, and `/datalake/archive/`. Verify their replication factors differ using `-stat %r`.

```
osboxes@osboxes:/usr/local/hadoop$ hdfs dfs -ls -R /datalake/
drwxr-xr-x   - osboxes supergroup          0 2026-02-19 08:16 /datalake/archive
-rw-r--r--   1 osboxes supergroup         16 2026-02-19 08:16 /datalake/archive/archive.txt
drwxr-xr-x   - osboxes supergroup          0 2026-02-19 07:30 /datalake/raw
drwxr-xr-x   - osboxes supergroup          0 2026-02-19 08:11 /datalake/raw/imaging
-rw-r--r--   3 osboxes supergroup         16 2026-02-19 08:11 /datalake/raw/imaging/xray.txt
drwxr-xr-x   - osboxes supergroup          0 2026-02-19 07:29 /datalake/raw/lab_results
drwxr-xr-x   - osboxes supergroup          0 2026-02-19 08:15 /datalake/raw/patients
-rw-r--r--   1 osboxes supergroup         15 2026-02-19 08:15 /datalake/raw/patients/patient.txt
drwxr-xr-x   - osboxes supergroup          0 2026-02-19 07:30 /datalake/raw/processed
```

Reflection 6.4

Why might a hospital want **different replication factors** for X-ray images (rep=3) vs. old archived logs (rep=1)? Consider: data criticality, storage cost, and recovery requirements.

A hospital may set a higher replication factor (rep=3) for X-ray images because they are critical for diagnosis and treatment, and losing them could seriously impact patient care. Higher replication increases availability and fault tolerance, so the data can still be accessed quickly even if a DataNode fails.

In contrast, old archived logs are less critical and are rarely accessed, so keeping only one replica (rep=1) reduces storage cost. Since these logs are not needed urgently, the hospital can accept lower redundancy and longer recovery time if a failure happens.

Part 7: Deliverables & Submission (10 min)

Submission Checklist

	Item	Description	Marks
	Screenshot 1	HDFS Web UI overview (Task 1.2)	1
	Task 2.1	HDFS commands for file operations	2
	Task 3.1	Block inspection results from `fsck`	2
	Task 3.2	Replication factor change + screenshot	2
	Task 4.1	Admin report values	1
	Task 4.2	Safe mode experiment + answers	2
	Task 5.1	Saudi data analysis answers	2
	Screenshot 5.2	HDFS directory structure	1
	Task 6.1	Small files experiment table + reflection	3
	Task 6.2	Fault tolerance observations + reflection	3
	Task 6.3	Block size table + reflection	3
	Task 6.4	Data lake structure + screenshot + reflection	3
		Total	**25**

Submission Instructions:

- Submit this completed lab document as a **PDF** with all screenshots, answers, and reflections.
- File name format: `CS4074_Lab2_YourName_YourID.pdf`
- Deadline: **One week from today's lab session.**
- Late submissions: **−5 marks per day.**

Before You Leave

```
# Stop HDFS when done
stop-dfs.sh

# Verify all daemons are stopped
jps
```

End of Lab 2

Next Lab: MapReduce Programming