MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DSE-3264 - Big Data Analytics
Laboratory Manual

Department	:	Data Science Engineering And Computer Applications
Course Name & code	:	DSE-3264 & Big Data Analytics Laboratory
Semester & branch	:	VI Sem & BTech Data Science & Engineering
Name of the faculty	:	Dr. Saraswati Koppad, Dr. Shavantrevva Sangappa Bilakeri

No of contact hours/week:

L	T	P	C
0	0	3	1

CONTENTS

Course Objectives ·

1. Gain practical experience using big data tools and platforms like Hadoop, Spark, Hive, Pig, and HBase to store, process, and analyze large datasets.
2. Implement Hadoop MapReduce for processing Big Data
3. Apply data analytics techniques to solve problems and extract meaningful insights using big data frameworks.

Course Outcomes:

At the end of this course, students will have the ability to

1. Demonstrate the ability to use big data frameworks such as Hadoop, Spark
2. Apply MapReduce techniques to process large data sets.
3. Demonstrate the ability to use big data tools such as Pig, Hive, and HBase to store and process Big Data
4. Apply analytical methods and techniques to solve data-driven problems.

Evaluation plan : (Tentative)

- Internal Assessment Marks: 60%
- End semester assessment of 2-hour duration: 40 %

Evaluation pattern

Internal Marks – 60 + End Sem - 40	
Internal Marks	Internal Assessment – 40 + Mid-sem - 20
Internal Assessment	Lab observations – 2*11(22) + Viva (18)
Lab Observation	Record (1*11) + Execution (1*11)
Viva	Quiz and mini-project (18)

INSTRUCTIONS TO THE STUDENTS

Pre-Lab Session Instructions

- Be on time, adhere to the institution's rules, and maintain decorum.
- Leave your mobile phones, pen drives, and other electronic devices in your bag and keep the bag in the designated place in the lab.
- Must Sign in to the log register provided.
- Make sure to occupy the allotted system and answer the attendance.

In-Lab Session Instructions

- Follow the instructions on the allotted exercises.
- Show the program and results to the instructors on completion of experiments.
- Copy the program and results for the lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the Exercises in Lab

- Academic honesty is required in all your work. You must solve all programming assignments independently, except where group work is authorized. This means you must not take, show, give, or otherwise allow others to take your program code, problem solutions, or other work.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.), give appropriate error messages, and suggest corrective actions.
    - Comments should be used to give the statement of the problem, and every function should indicate the purpose of the function, inputs, and outputs.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.
    - Make use of constants and type definitions wherever needed.
    - The exercises for each week are divided into three sets:
        - Solved solutions
        - Lab exercises - to be completed during lab hours
        - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill

Questions for lab tests and examinations are not necessarily limited to the questions in the manual but may involve some variations and/or combinations of the questions.

THE STUDENTS SHOULD NOT

- Possess mobile phones or any other electronic gadgets during lab hours.
- Go out of the lab without permission.
- **Change/update any configuration in your allotted system. If so, the student will lose internal assessment marks**

Week 1 - Understanding Hadoop and HDFS Basic Commands

Introduction to the Hadoop Ecosystem and HDFS

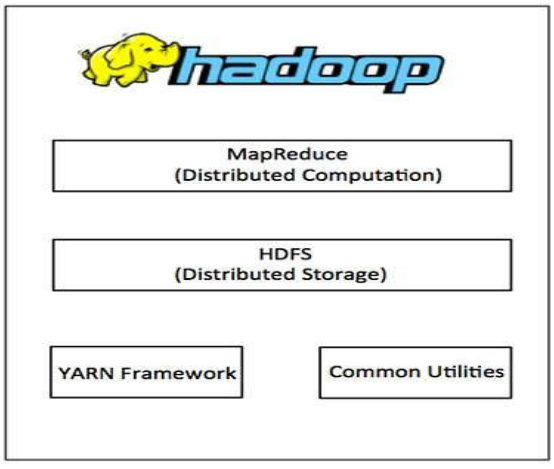
The Hadoop Ecosystem is a collection of tools and frameworks that work together to manage and process big data. HDFS (Hadoop Distributed File System) is the backbone of the Hadoop ecosystem, providing distributed storage and fault tolerance.

Hadoop Ecosystem Overview

The Hadoop Ecosystem is built around the core Hadoop components and includes tools for data storage, processing, querying, and analytics. Here's a breakdown:

Core Components:

1. **HDFS (Hadoop Distributed File System):**
   - Provides distributed storage.
   - Handles large files across multiple machines.
2. **YARN (Yet Another Resource Negotiator):**
   - Manages resources and job scheduling in the cluster.
3. **MapReduce:**
   - Programming model for data processing.



Supporting Tools:

1. **Apache Hive:**
    - Data warehouse tool for querying data in HDFS using SQL-like language (HiveQL).
2. **Apache HBase:**
    - NoSQL database for real-time read/write access to large datasets.
3. **Apache Spark:**
    - Fast, in-memory data processing engine.
4. **Apache Pig:**
    - High-level platform for creating MapReduce programs using a scripting language (Pig Latin).
5. **Apache Sqoop:**
    - Tool for transferring data between Hadoop and relational databases.
6. **Apache Flume:**
    - Tool for collecting and transferring large amounts of log data into HDFS.
7. **ZooKeeper:**
    - Centralized service for managing distributed systems.
8. **Oozie:**
    - Workflow scheduler for Hadoop jobs.

HDFS Basics

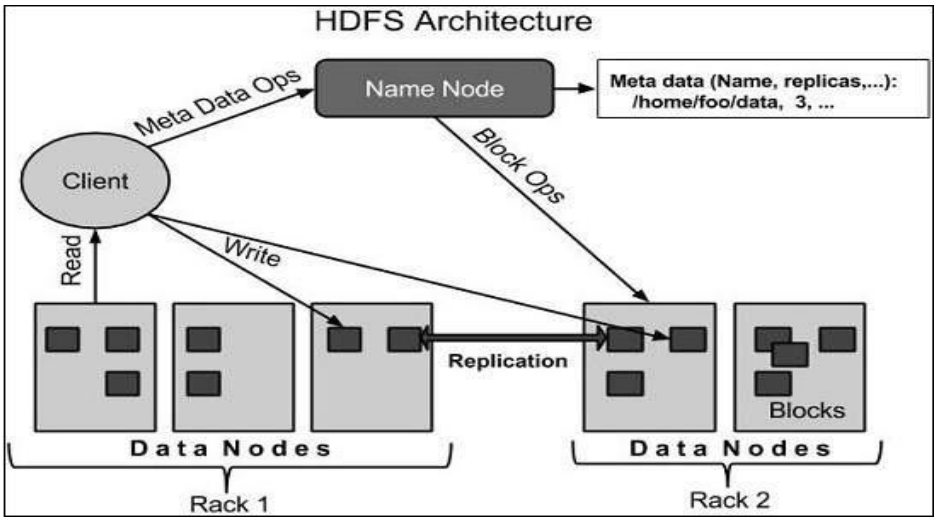
HDFS is the storage layer of Hadoop, designed for scalability and fault tolerance. It handles massive data volumes by breaking files into smaller chunks and distributing them across a cluster.

HDFS Features:

- Replication: Ensures fault tolerance by replicating data blocks (default: 3 copies).
- Write-Once, Read-Many: Optimized for batch processing.
- Scalability: Handles petabytes of data across thousands of nodes.

HDFS Architecture:

1. NameNode:
    - Maintains metadata (e.g., file structure, permissions).
    - Coordinates DataNodes but does not store data.
2. DataNodes:
    - Store actual data blocks.
    - Perform read/write operations as instructed by NameNode.
3. Secondary NameNode:
    - Periodically saves snapshots of NameNode metadata (not a backup).



Working with Hadoop:

- Open a new terminal and start the Hadoop service by following commands

```
start-dfs.sh
start-yarn.sh
```

This command initializes all the required Hadoop daemons for the cluster to become operational.

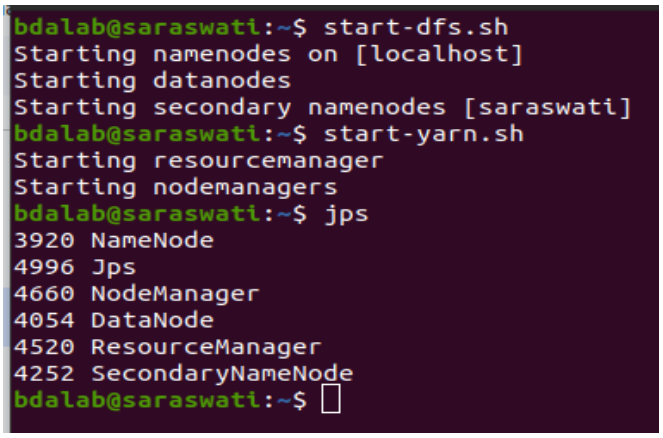
When you want to stop the services use the command:

```
stop-dfs.sh
stop-yarn.sh
```

- Check the Status of Hadoop Services (To confirm that all necessary Hadoop services are up and running, you can check their status using the 'jps' command)

```
jps
```

The 'jps' command displays the Java Virtual Machine (JVM) processes and should show a list of Hadoop services running, such as the NameNode, DataNode, ResourceManager, and NodeManager.



- **Accessing Hadoop Namenode and Resource Manager:**

  To access the Hadoop Namenode, open a web browser and enter the following URL:

  http://localhost:9870

  Similarly, to access the Hadoop Resource Manager, open a web browser and enter the following URL:

  http://localhost:8088

- **Verifying the Hadoop Cluster:**

  You can check the Hadoop version by following command

  ```
  hadoop version
  ```

  Create Directories in HDFS:

  ```
  hdfs dfs -mkdir /user/dir_name
  ```

  List Directories in HDFS:

  ```
  hdfs dfs -ls /
  hdfs dfs -ls /user/
  ```

  Transfer Files to the Hadoop File System:

  ```
  hdfs dfs -put source-filename /destination_folder/
  ```

Week-1 Exercise:

1. Practice Linux commands (No need to write in the record book)
    a. What command would you use to display the current directory?
    b. How do you list all files, including hidden ones, in a directory?
    c. Which command is used to create a new directory?
    d. How can you check the currently logged-in user?
    e. What does the pwd command do?
    f. How do you copy a file from one location to another?
    g. What is the command to move a file?
    h. How can you delete a directory and all its contents?
    i. How do you rename a file in Linux?
    j. What is the difference between `rm` and `rmdir`?
    k. What command is used to change the permissions of a file?
    l. How can you view the permissions of a file?
    m. Which command changes the owner of a file?
    n. How do you add execute permission for the owner of a file?
    o. What is the command to display all currently running processes?
    p. How can you check the IP address of your system?
    q. How do you display the contents of a file?
    r. How can you display the current date and time in Linux?

2. Explore Basic HDFS Commands
    a. Remove directory
    b. View/Read File Contents
    c. Download/copy a file from HDFS to the local system
    d. Copy/move files within HDFS
    e. Remove file from HDFS
    f. View file permission

Week 2: Advanced HDFS Commands and MapReduce

Week 2 Exercises (Advanced HDFS Commands):

Create a directory in Hadoop (/user/your_reg_no/lab2). Use the same directory to keep all your files.

hdfs dfs -mkdir /user/your_reg_no/lab2

(a).

1. Upload a new file (Ex. File1.txt) into Hadoop. Demonstrate the hdfs commands for the following:
    a. Check the permission of the file
    b. Change the permission of the file.
    c. Check ownership of the file
2. Upload a large file (Ex: largefile.txt) into Hadoop. Demonstrate the hdfs commands for following.
    a. Check block details of the file
    b. View the current replication factor for file
    c. Modify the replication factor of the file
3. Demonstrate hdfs commands for the following tasks
    a. Check directory size for the directory you have created (Disk usage analysis)
    b. Find the total capacity and usage of HDFS
4. Upload a text file with sample data to the Hadoop (Ex: analytics.txt). Using HDFS commands, find
    a. Number of lines in the file
    b. Number of occurrences of a specific word (Ex. Hadoop) in the file.

(b). Executing simple MapReduce jobs:

What is Hadoop MapReduce?

Hadoop MapReduce is a programming model and processing framework for processing large datasets in a distributed environment. It divides the job into smaller tasks, processes them in parallel, and consolidates the results.

Key Components

1. **Map Phase:**
    - Processes input data and generates intermediate key-value pairs.
    - Example: Counting words in a document; the mapper emits each word as a key and the count 1 as a value.
2. **Shuffle and Sort Phase:**
    - Intermediate key-value pairs are grouped and sorted by key.
    - Ensures all values for a specific key are brought together.
3. **Reduce Phase:**
    - Processes grouped data from the shuffle phase to generate final output.
    - Example: Aggregating word counts.

MapReduce Workflow

1. **Input Data:**
    - Stored in HDFS and divided into splits.
2. **Mapper:**
    - Processes each split and outputs key-value pairs.
3. **Shuffle and Sort:**
    - Intermediate outputs are shuffled and sorted by the framework.
4. **Reducer:**
    - Processes sorted key-value pairs and generates the final result.
5. **Output:**
    - Written back to HDFS.

(b). Executing sample MapReduce jobs:

What is Hadoop MapReduce?

Hadoop MapReduce is a programming model and processing framework for processing large datasets in a distributed environment. It divides the job into smaller tasks, processes them in parallel, and consolidates the results.

Key Components

4. **Map Phase:**
    - Processes input data and generates intermediate key-value pairs.

- Example: Counting words in a document; the mapper emits each word as a key and the count 1 as a value.

5. **Shuffle and Sort Phase:**
    - Intermediate key-value pairs are grouped and sorted by key.
    - Ensures all values for a specific key are brought together.
6. **Reduce Phase:**
    - Processes grouped data from the shuffle phase to generate final output.
    - Example: Aggregating word counts.

MapReduce Workflow

6. **Input Data:**
    - Stored in HDFS and divided into splits.
7. **Mapper:**
    - Processes each split and outputs key-value pairs.
8. **Shuffle and Sort:**
    - Intermediate outputs are shuffled and sorted by the framework.
9. **Reducer:**
    - Processes sorted key-value pairs and generates the final result.
10. **Output:**
    - Written back to HDFS.

Part 1: Run bellow commands to install python (Locally)

wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz
tar -xvf Python-3.6.5.tgz

Step-1. Write a Mapper

Mapper.py: Initially the partition of content takes place based on line.split() function, number of partitions made = number of mapper class gets created. Mapper overrides the ―map‖ function which provides <key, value> pairs as the input. Even the key is repeated in same or different mapper class it doesnot matter as the default value for every key is assigned to be as "1". A Mapper implementation may output<key,value> pairs using the provided Context .

Input value of the WordCount Map task will be a line of text from the input data file and the key would be the line number <line_number, line_of_text> . Map task outputs <word, one> for each word in the line of text.

Pseudo-code : mapper.py

```
#!/usr/bin/python3
"mapper.py"
import sys
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        print ('%s\t%s' % (word, 1))
```

Step-2. Write a Reducer

A Reducer collects the intermediate <key,value> output from multiple map tasks and assemble a single result. Here, the WordCount program will sum up the occurrence of each word to pairs as <word, occurrence>.

Pseudo-code: reducer.py

```
#!/usr/bin/python3
"reducer.py"
import sys
current_word = None
current_count = 0
for line in sys.stdin:
        line = line.strip()
        word, count = line.split('\t')
        count = int(count)
        if current_word == word:
                current_count += count
        else:
                if current_word:
                        print ('%s\t%s' % (current_word, current_count))
                current_word = word
                current_count = 1
```

TO run word count program locally use following commands

Command: cat input.txt |pyhton3 mapper.py

Output:

hi 1
how 1
are 1
you 1
i 1
am 1
good 1
hope 1
you 1
doing 1
good 1
too 1
how 1
about 1
you. 1
i 1
am 1
in 1
manipal 1
studying 1
Btech 1
in 1
Data 1
science. 1

Command: cat input.txt |python3 mapper.py|sort|python3 reducer.py

Output:

about 1
am 2
are 1

Btech 1
Data 1
doing 1
good 2
hi 1
hope 1
how 2
i 2
in 2
manipal 1
science. 1
studying 1
too 1
you 2
you. 1

Part 2. TO run word count program on Hadoop framework use following command:

(i)Prepare input data (inputwc.txt)

Load input.txt into Hadoop:

hdfs dfs -put inputwc.txt /user/studentregno/lab2

(ii) Run the Example MapReduce Job: (Available with Hadoop Installation)

hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.1.jar wordcount /user/studentregno/lab2/inputwc.txt/user/studentregno/lab2/output

(iii) View the Results

hdfs dfs -cat /user/studentregno/lab2/output/part-r-00000

Week 2: Exercise: MapReduce with Python

1. Consider the text file (consider larger file size) of your choice and perform word count using MapReduce technique.
2. Perform Matrix operations using MapReduce by considering 3 * 3 matrix and perform following operations:
    i. Matrix addition and subtraction
    ii. Matrix Multiplication
    iii. Matrix transpose

Note: Consider 3*3 matrix content as shown below

a,0,0,10
a,0,1,20
a,0,2,30
a,1,0,40
a,1,1,50
a,1,2,60
a,2,0,70
a,2,1,80
a,2,2,90

b,0,0,1
b,0,1,2
b,0,2,3
b,1,0,4
b,1,1,5
b,1,2,6
b,2,0,7
b,2,1,8
b,2,2,9

3. Create a text file containing the 20 student details such as registration number, name and marks (ex: 1001, john,45) .Write a MapReduce program to sort data by student name

Week 3: MapReduce Programs

Example Program: (calculate word counts in a text file)

(i) Create a mapper script

The Mapper processes the input line by line, splitting it into words and emitting each word with a count of 1.

```
#!/usr/bin/env python3
# mapper.py
import sys

# Read lines from standard input
for line in sys.stdin:
    # Split the line into words
    words = line.strip().split()
    for word in words:
        # Emit each word with a count of 1
```

(ii) Create a Reducer script

The Reducer receives sorted input from the Mapper, groups values by key, and computes the sum of counts for each word.

```
#!/usr/bin/env python3
# reducer.py
import sys

current_word = None
current_count = 0
word = None

# Read lines from standard input
for line in sys.stdin:
    # Parse the input we got from mapper
    word, count = line.strip().split("\t")
    count = int(count)

    # If this is the first word or a new word
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # Emit the word and its count
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

# Emit the last word
if current_word == word:
    print(f"{current_word}\t{current_count}")
```

(iii) Prepare input data (inputwc.txt)

Manipal Institute of Technology Manipal
Manipal Academy of Higher Education
Department of Data Science and Computer Application
Manipal, Karnataka, India

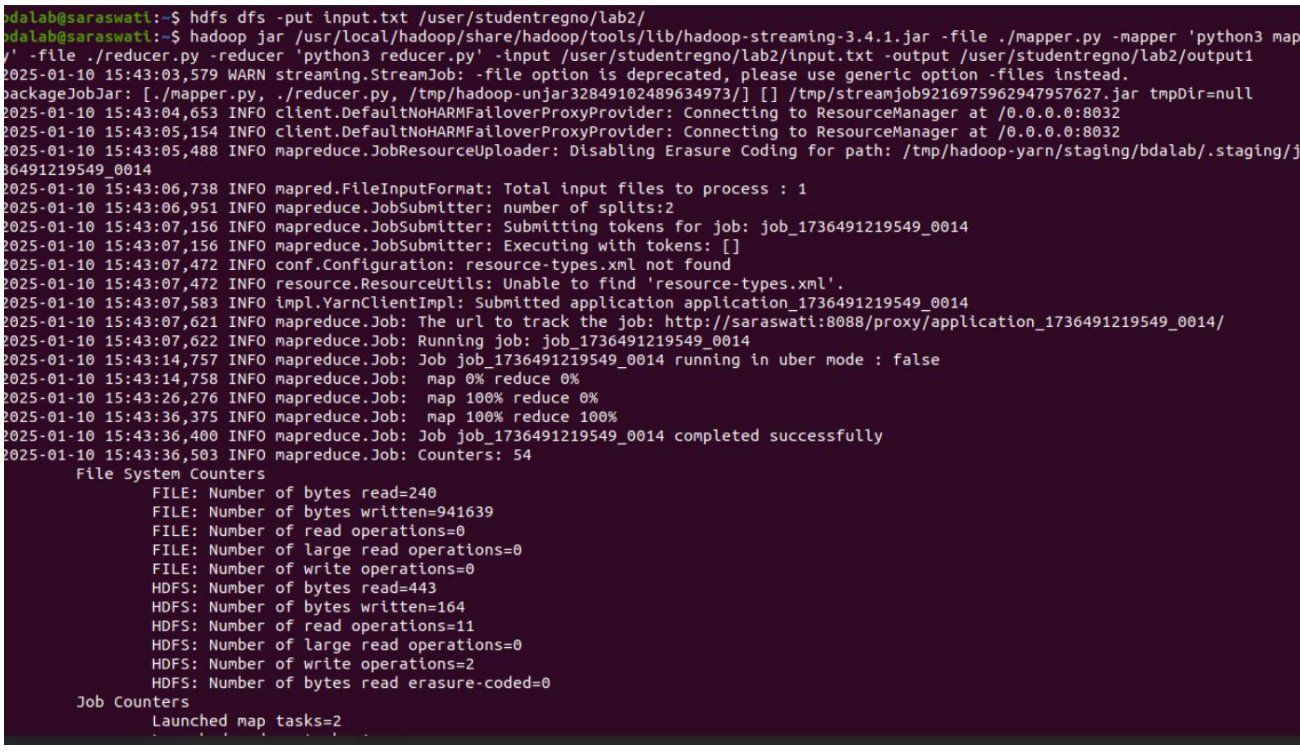
Load input.txt into Hadoop:

hdfs dfs -put inputwc.txt /user/studentregno/lab2

(iv) Run the MapReduce Job:

Execute the job using Hadoop Streaming:

```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./mapper.py -mapper 'python3 mapper.py' -file ./reducer.py -reducer 'python3 reducer.py' -input /user/studentregno/lab2/input.txt -output /user/studentregno/lab2/output1
```



```
bdalab@saraswati:~$ hdfs dfs -put input.txt /user/studentregno/lab2/
bdalab@saraswati:~$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./mapper.py -mapper 'python3 map
' -file ./reducer.py -reducer 'python3 reducer.py' -input /user/studentregno/lab2/input.txt -output /user/studentregno/lab2/output1
2025-01-10 15:43:03,579 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./mapper.py, ./reducer.py, /tmp/hadoop-unjar32849102489634973/] [] /tmp/streamjob9216975962947957627.jar tmpDir=null
2025-01-10 15:43:04,653 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-10 15:43:05,154 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-10 15:43:05,488 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/bdalab/.staging/j
36491219549_0014
2025-01-10 15:43:06,738 INFO mapred.FileInputFormat: Total input files to process : 1
2025-01-10 15:43:06,951 INFO mapreduce.JobSubmitter: number of splits:2
2025-01-10 15:43:07,156 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1736491219549_0014
2025-01-10 15:43:07,156 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-01-10 15:43:07,472 INFO conf.Configuration: resource-types.xml not found
2025-01-10 15:43:07,472 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-01-10 15:43:07,583 INFO impl.YarnClientImpl: Submitted application application_1736491219549_0014
2025-01-10 15:43:07,621 INFO mapreduce.Job: The url to track the job: http://saraswati:8088/proxy/application_1736491219549_0014/
2025-01-10 15:43:07,622 INFO mapreduce.Job: Running job: job_1736491219549_0014
2025-01-10 15:43:14,757 INFO mapreduce.Job: Job job_1736491219549_0014 running in uber mode : false
2025-01-10 15:43:14,758 INFO mapreduce.Job:  map 0% reduce 0%
2025-01-10 15:43:26,276 INFO mapreduce.Job:  map 100% reduce 0%
2025-01-10 15:43:36,375 INFO mapreduce.Job:  map 100% reduce 100%
2025-01-10 15:43:36,400 INFO mapreduce.Job: Job job_1736491219549_0014 completed successfully
2025-01-10 15:43:36,503 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=240
                FILE: Number of bytes written=941639
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=443
                HDFS: Number of bytes written=164
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=2
```

(v) View the Results

hdfs dfs -cat /user/studentregno/lab2/output1/part-00000

```
                CPU time spent (ms)=7270
                Physical memory (bytes) snapshot=1156423680
                Virtual memory (bytes) snapshot=7669239808
                Total committed heap usage (bytes)=1174929408
                Peak Map Physical memory (bytes)=468488192
                Peak Map Virtual memory (bytes)=2556018688
                Peak Reduce Physical memory (bytes)=222208000
                Peak Reduce Virtual memory (bytes)=2557878272
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=231
        File Output Format Counters
                Bytes Written=164
2025-01-10 15:43:36,503 INFO streaming.StreamJob: Output directory: /user/studentregno/lab2/output1
bdalab@saraswati:~$ hdfs dfs -cat /user/studentregno/lab2/output1/part-00000
Academy 1
Application     1
Computer        1
Data    1
Department      1
Education       1
Higher  1
India   1
Institute       1
Karnataka,      1
Manipal 3
Manipal,        1
Science 1
Technology      1
and     1
of      3
bdalab@saraswati:~$
```

Week-3 Exercise:

1. Write a Hadoop MapReduce program to count the frequency of each character in a text file.
2. Write a Hadoop MapReduce program to find the maximum temperature for each year in a weather dataset.
3. Write a Hadoop MapReduce program to find the top N records (e.g., top 3 highest scores) from a dataset.
4. Write a Hadoop MapReduce program to calculate the average value for each key in a dataset

Week-4 Advanced Programs on MapReduce

Solved Program: Implement a MapReduce program to perform secondary sorting, where records are sorted by a primary and secondary key.

Write a MapReduce program to Sort a dataset of employee records by department (primary key) and salary (secondary key) in ascending order.

Sample Input:

1,HR,Alice,7000

2,IT,Bob,9000

3,HR,Charlie,8000

4,IT,Dave,9500

Expected Output:

HR,Alice,7000

HR,Charlie,8000

IT,Bob,9000

IT,Dave,9500

Solution:

In this program, we'll sort employee records by department (primary key) and salary (secondary key) in ascending order.

Step-1 : Write mapper and reducer program as follows:

Mapper (4s_mapper.py):

- Emits the department and salary as the key.
- Emits the remaining details (employee ID and name) as the value.
- Sorting is achieved by Hadoop's built-in sorting mechanism, which sorts by key during the shuffle and sort phase.

```python
#!/usr/bin/env python3
import sys

# Read input line by line
for line in sys.stdin:
    # Strip leading/trailing whitespace
    line = line.strip()

    # Split the line into fields
    fields = line.split(',')

    # Ensure the input format is valid
    if len(fields) == 4:
        employee_id, department, name, salary = fields

        # Emit department and salary as composite key, with other details as
        print(f"{department}\t{salary}\t{employee_id},{name}")
```

Reducer (4s_reducer.py):

- Group records by department.
- Sort records within each department by salary using Python's sorted function.

```python
#!/usr/bin/env python3
import sys
from collections import defaultdict

# Dictionary to store department data
department_data = defaultdict(list)

# Process each line of input
for line in sys.stdin:
    # Strip leading/trailing whitespace
    line = line.strip()

    # Split the line into department, salary, and other details
    try:
        department, salary, details = line.split('\t', 2)
        salary = int(salary)  # Convert salary to integer for sorting
        department_data[department].append((salary, details))
    except ValueError:
        continue

# Iterate over each department and sort by salary
for department, records in department_data.items():
    # Sort records by salary
    sorted_records = sorted(records, key=lambda x: x[0])

    # Emit sorted records
    for salary, details in sorted_records:
        print(f"{department},{details}")
```

Step-2: Create input data (4s_input.txt) and copy it into HDFS:

```
1,HR,Eve,6000
2,HR,Alice,7000
3,HR,Charlie,8000
4,IT,Bob,9000
5,IT,Dave,9500
```

To load file on Hadoop:

```
hdfs dfs -mkdir /user/studentregno/lab4
hdfs dfs -put 4s_input.txt /user/studentregno/lab4
```

Step-3: Run the program on Hadoop

Run the MapReduce Job:

Execute the job using Hadoop Streaming:

```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./4s_mapper.py -mapper 'python3 4s_mapper.py' -file ./4s_reducer.py -reducer 'python3 4s_reducer.py' -input /user/studentregno/lab4/4s_input.txt -output /user/studentregno/lab4/output1
```

```
bdalab@saraswati:~$ hdfs dfs -put 4s_input.txt /user/studentregno/lab4
bdalab@saraswati:~$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./4s_mapper.py -mapper 'python3 4s_mapper.py' -file ./4s_reducer.py -reducer 'python3 4s_reducer.py' -input /user/studentregno/lab4/4s_input.txt -output /user/studentregno/lab4/output2
2025-01-23 12:58:36,542 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./4s_mapper.py, ./4s_reducer.py, /tmp/hadoop-unjar3773630761779343200/] [] /tmp/streamjob562498201783300010.jar tmpDir=null
2025-01-23 12:58:37,453 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-23 12:58:37,714 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-01-23 12:58:38,683 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/bdalab/.staging/job_1737616633980_0002
2025-01-23 12:58:42,754 INFO mapred.FileInputFormat: Total input files to process : 1
2025-01-23 12:58:43,986 INFO mapreduce.JobSubmitter: number of splits:2
2025-01-23 12:58:44,813 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1737616633980_0002
2025-01-23 12:58:44,813 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-01-23 12:58:45,027 INFO conf.Configuration: resource-types.xml not found
2025-01-23 12:58:45,027 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-01-23 12:58:45,102 INFO impl.YarnClientImpl: Submitted application application_1737616633980_0002
2025-01-23 12:58:45,136 INFO mapreduce.Job: The url to track the job: http://saraswati:8088/proxy/application_1737616633980_0002/
2025-01-23 12:58:45,137 INFO mapreduce.Job: Running job: job_1737616633980_0002
2025-01-23 12:58:52,322 INFO mapreduce.Job: Job job_1737616633980_0002 running in uber mode : false
2025-01-23 12:58:52,325 INFO mapreduce.Job:  map 0% reduce 0%
2025-01-23 12:58:58,524 INFO mapreduce.Job:  map 100% reduce 0%
2025-01-23 12:59:03,571 INFO mapreduce.Job:  map 100% reduce 100%
2025-01-23 12:59:06,642 INFO mapreduce.Job: Job job_1737616633980_0002 completed successfully
2025-01-23 12:59:06,751 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=93
                FILE: Number of bytes written=941420
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=341
                HDFS: Number of bytes written=57
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
```

```
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=99
                Reduce input records=5
                Reduce output records=5
                Spilled Records=10
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=675
                CPU time spent (ms)=3100
                Physical memory (bytes) snapshot=1157881856
                Virtual memory (bytes) snapshot=7671750656
                Total committed heap usage (bytes)=1189609472
                Peak Map Physical memory (bytes)=473272320
                Peak Map Virtual memory (bytes)=2557829120
                Peak Reduce Physical memory (bytes)=213995520
                Peak Reduce Virtual memory (bytes)=2557796352
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=123
        File Output Format Counters
                Bytes Written=57
2025-01-23 12:59:06,751 INFO streaming.StreamJob: Output directory: /user/studentregno/lab4/output2
bdalab@saraswati:~$ hdfs dfs -cat /user/studentregno/lab4/output2/part-00000
HR,1,Eve
HR,2,Alice
HR,3,Charlie
IT,4,Bob
IT,5,Dave
bdalab@saraswati:~$
```

Week-4 Exercise:

- Write a Hadoop MapReduce program to perform a **reduce-side join** to merge two datasets based on a common key.

  Hint: Join two datasets:

  Students dataset:

  StudentID,Name,CourseID

  Courses dataset:

  CourseID,CourseName,Sem

  Perform an **inner join** to output:

  StudentID,Name,CourseName,Sem

- Write a Hadoop MapReduce program to implement a distributed **word count** program

that filters out stop words during processing.

Hint: Count the frequency of each word in a text dataset while ignoring stop words (e.g., "is," "the," "and").

Sample Input:

Doc1 This is a sample document

Doc2 Another document with sample text

Expected Output:

Another 1

Document 2

Sample 2

Text 1

- Write a Hadoop MapReduce program to create an **inverted index** that maps each word to the list of documents in which it appears.

  Hint: Generate an inverted index for a collection of documents.

  Sample Input:

  Doc1 Hadoop MapReduce is powerful

  Doc2 MapReduce is scalable

  Doc3 Hadoop and Spark are popular

  Expected Output:

  Hadoop Doc1,Doc3

  MapReduce Doc1,Doc2

  Powerful Doc1

  Scalable Doc2

  Spark Doc3

  Popular Doc3