



## **DSE-3264 - Big Data Analytics**

### **Laboratory Manual**

<b>Department</b>	:	Data Science Engineering And Computer Applications								
<b>Course Name &amp; code</b>	:	DSE-3264 & Big Data Analytics Laboratory								
<b>Semester &amp; branch</b>	:	VI Sem & BTech Data Science & Engineering								
<b>Name of the faculty</b>	:	Dr. Saraswati Koppad, Dr. Shavantrevva Sangappa Bilakeri								
<b>No of contact hours/week:</b>	:	<table border="1"><tr><td><b>L</b></td><td><b>T</b></td><td><b>P</b></td><td><b>C</b></td></tr><tr><td>0</td><td>0</td><td>3</td><td>1</td></tr></table>	<b>L</b>	<b>T</b>	<b>P</b>	<b>C</b>	0	0	3	1
<b>L</b>	<b>T</b>	<b>P</b>	<b>C</b>							
0	0	3	1							

## CONTENTS

Lab No.	Title	Page No
	Course Objectives	
	Evaluation Plan	
	Instructions to Students	
1	Understanding Hadoop with HDFS Basic Commands	
2	Explore advanced HDFS operations	
3	Map Reduce basics	
4	Advanced Map reduce	
5	Exploring Pig	
6	Exploring Hive	
7	Exploring HBase	
8	Spark Basics	
9	Exploring SparkSQL/SparkML	
10	Data analytics using Spark.	
11	Data analytics using Spark	
12	End Sem exam	

**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**

<b>Internal Marks – 60 + End Sem - 40</b>	
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**

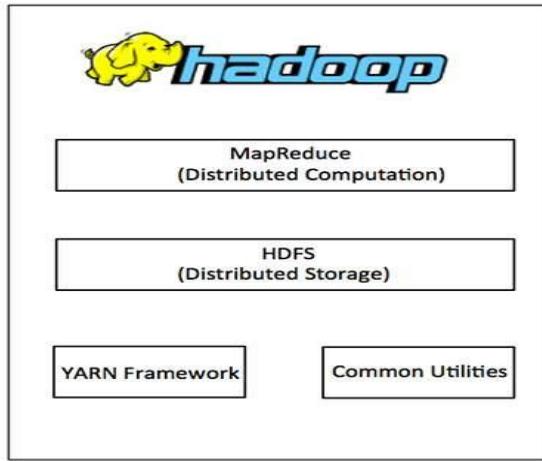
The Hadoop Ecosystem is a collection of tools and frameworks that work together to manage and process big data. HDFS (Hadoop Distributed File System) is the backbone of the Hadoop ecosystem, providing distributed storage and fault tolerance.

### **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.

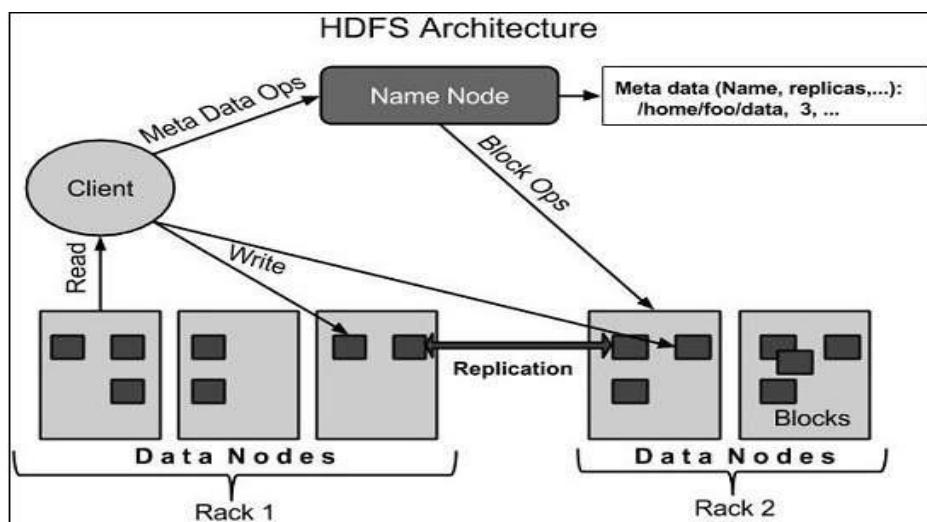
### **HDFS Features:**

- 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.

```
[bdalab@saraswati:~$ start-dfs.sh  
Starting namenodes on [localhost]  
Starting datanodes  
Starting secondary namenodes [saraswati]  
[bdalab@saraswati:~$ start-yarn.sh  
Starting resourcemanager  
Starting nodemanagers  
[bdalab@saraswati:~$ jps  
3920 NameNode  
4996 Jps  
4660 NodeManager  
4054 DataNode  
4520 ResourceManager  
4252 SecondaryNameNode  
[bdalab@saraswati:~$ ]
```

- **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 below 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 does not 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

```

```

dalab@saraswati:~$ hdfs dfs -put input.txt /user/studentregno/lab2/
dalab@saraswati:~$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.4.1.jar -file ./mapper.py -mapper 'python3
' -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.

```

#!/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 value
        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.

```

#!/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/streamjob56249820178330010.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

## Week-5 Exploring Pig

### What is Apache Pig?

Apache Pig is a high-level data flow scripting language used to process large datasets in Hadoop. It provides an abstraction over MapReduce, allowing users to write simpler scripts instead of complex Java code. Pig uses its own scripting language called Pig Latin, which is similar to SQL but more flexible for data transformations and ETL (Extract, Transform, Load) operations.

### Key Features of Apache Pig

- Ease of Use – Uses a simple, SQL-like language (Pig Latin).
- Abstraction Over MapReduce – No need to write complex Java code.
- Scalability – Runs efficiently on large datasets.
- Extensibility – Supports User-Defined Functions (UDFs) in Python and Java.
- Flexibility – Handles both structured and semi-structured data.

### Apache Pig vs. MapReduce

Feature	Apache Pig	MapReduce
Language	Pig Latin	Java
Complexity	Simple	Complex
Use Case	ETL, data transformation	Custom logic
Performance	Faster than Hive	Slower

### Pig Architecture

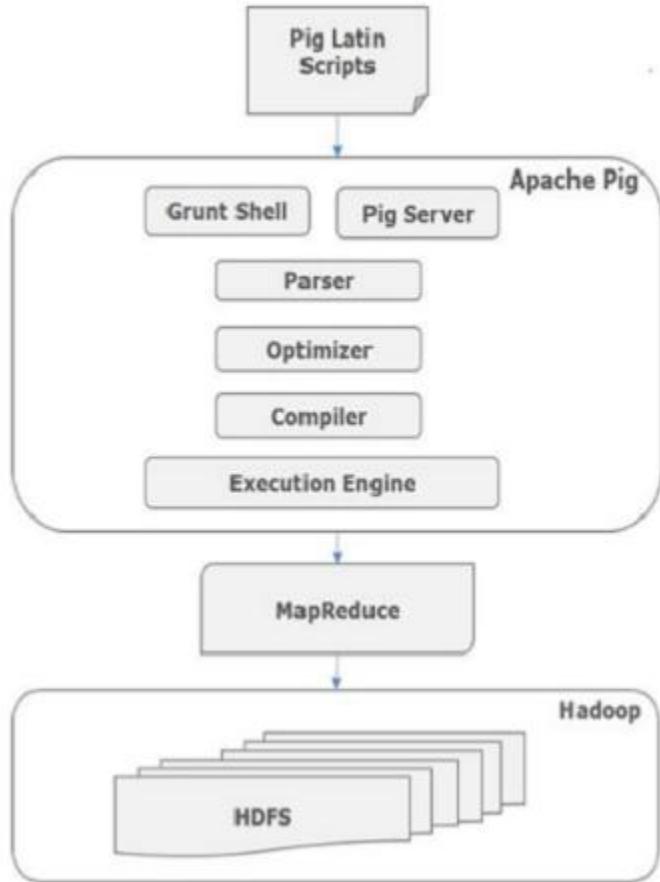
Apache Pig consists of:

- Pig Latin Interpreter – Parses and converts scripts into MapReduce jobs.
- Grunt Shell – Interactive command-line interface to run Pig scripts.
- Execution Engine – Converts scripts into optimized execution plans.
- HDFS / Local Mode – Works on Hadoop HDFS or standalone mode.

Modes of Execution:

- Local Mode – Runs without Hadoop (pig -x local).

- MapReduce Mode – Runs on Hadoop (pig -x mapreduce).



### When to Use Apache Pig?

- ETL (Extract, Transform, Load) workflows
- Data preprocessing before analysis
- Handling semi-structured data like logs, JSON, CSV
- Quick prototyping compared to MapReduce

Apache Pig is widely used in big data processing pipelines at companies like Yahoo, Twitter, and LinkedIn.

## Solved Exercise: First Pig Script: Word Count

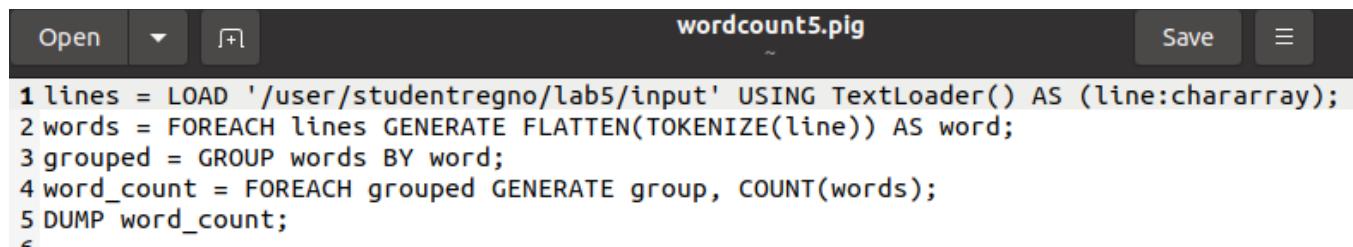
Step 1: Create a Sample Input File (input.txt)

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

Load it into hdfs:

```
hdfs dfs -mkdir /user/studentregno/lab5  
hdfs dfs -put input.txt /user /studentregno/lab5
```

Step 2: Write Pig Latin Script and save it as wordcount5.pig



The screenshot shows a code editor window with the following details:

- File name: wordcount5.pig
- Content:

```
1 lines = LOAD '/user/studentregno/lab5/input' USING TextLoader() AS (line:chararray);  
2 words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word;  
3 grouped = GROUP words BY word;  
4 word_count = FOREACH grouped GENERATE group, COUNT(words);  
5 DUMP word_count;
```
- Toolbar buttons: Open, Save, and others.

Step 3:

Run the Script

Switch to Pig bin directory

```
cd /usr/local/pig/bin
```

To avoid re-connecting to server, run the following command

```
rm-jobhistory-daemon.sh start historyserver
```

Run the script with following command

```
./pig -x mapreduce /home/bdalab/wordcount5.pig
```

```

bdalab@saraswati:/usr/local/pig/bin$ ./pig -x mapreduce /home/bdalab/wordcount5.pig
2025-01-31 08:50:56,059 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2025-01-31 08:50:56,061 INFO pig.ExecTypeProvider: Picked MAPREDUCE as the ExecType
2025-01-31 08:50:56,116 [main] INFO org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2025-01-31 08:50:56,116 [main] INFO org.apache.pig.Main - Logging error messages to: /usr/local/pig/bin/pig_1738293656109.log
2025-01-31 08:50:56,464 [main] INFO org.apache.pig.impl.util.Utils - Default bootstrap file /home/bdalab/.pigbootup not found
2025-01-31 08:50:56,536 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2025-01-31 08:50:56,536 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://127.0.0.1:9000/
2025-01-31 08:50:57,142 [main] INFO org.apache.pig.PigServer - Pig Script ID for the session: PIG-wordcount5.pig-3a570a87-8288-4936-a0ad-022995219d85
2025-01-31 08:50:57,142 [main] WARN org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
2025-01-31 08:50:57,792 [main] INFO org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of size 699400192 to monitor . collectionUsageThreshold = 489580128, usageThreshold = 489580128
2025-01-31 08:50:57,962 [main] INFO org.apache.tools.pigstats.ScriptState - Pig features used in the script: GROUP_BY
2025-01-31 08:50:57,989 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 08:50:58,026 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyP rune, Constantcalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NestedLimitOptimizer, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
2025-01-31 08:50:58,136 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false
2025-01-31 08:50:58,153 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.CombinerOptimizerUtil - Choosing to move algebraic for each to combiner
2025-01-31 08:50:58,183 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1
2025-01-31 08:50:58,183 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1
2025-01-31 08:50:58,339 [main] INFO org.apache.hadoop.yarn.client.DefaultNoHARMFailoverProxyProvider - Connecting to ResourceManager at /0.0.0.0:8032
2025-01-31 08:50:58,792 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2025-01-31 08:50:58,816 [main] INFO org.apache.tools.pigstats.mapreduce.MRScriptState - Pig script settings are added to the job
2025-01-31 08:50:58,828 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.reduce.markreset.buffer.percent is deprecated. Instead, use mapreduce.reduce.markreset.buffer.percent
2025-01-31 08:50:58,828 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markreset.buffer.percent is not set, set to default 0.3
2025-01-31 08:50:58,828 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Input paths to process : 1
2025-01-31 08:53:59,140 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-01-31 08:53:59,152 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2025-01-31 08:53:59,157 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2025-01-31 08:53:59,196 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-01-31 08:53:59,196 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(of,3)
(and,1)
(Data,1)
(India,1)
(Higher,1)
(Academy,1)
(Manipal,4)
(Science,1)
(Computer,1)
(Education,1)
(Institute,1)
(Karnataka,1)
(Department,1)
(Technology,1)
(Application,1)
2025-01-31 08:53:59,266 [main] INFO org.apache.pig.Main - Pig script completed in 3 minutes, 3 seconds and 352 milliseconds (183352 ms)
bdalab@saraswati:/usr/local/pig/bin$ 

```

## Working with Datasets in Pig:

### Step-1: Loading Data into Pig:

Create a .csv file (employee.csv)

101,John,30,HR,50000
102,Alice,25,IT,60000
103,Bob,35,Finance,70000
104,Joy,36,Manager,55000

Load it on HDFS

```
hdfs dfs -put employee.csv /user/studentregno/lab5
```

Step 2: Write Pig Latin Script and save it as loadcsv.pig

```
1 employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')  
2 AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);  
3 DUMP employees;
```

Step-3: Run the script with following command

Loading csv:

```
./pig -x mapreduce /home/bdalab/loadcsv.pig
```

```
2025-01-31 09:07:20,279 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2025-01-31 09:07:20,326 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled  
is deprecated. Instead, use yarn.system-metrics-publisher.enabled  
2025-01-31 09:07:20,329 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.  
2025-01-31 09:07:20,359 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1  
2025-01-31 09:07:20,359 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(101,John,30,HR,50000)  
(102,Alice,25,IT,60000)  
(103,Bob,35,Finance,70000)  
(104,Joy,36,Manager,55000)  
2025-01-31 09:07:20,517 [main] INFO org.apache.pig.Main - Pig script completed in 3 minutes, 8 seconds and 480 milliseconds (188480 ms)  
bdalab@saraswati:/usr/local/pig/bin$
```

**Filtering Data: Find employees with a salary greater than 60,000.**

Write Pig Latin Script and save it as filter.pig

```
1 employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')  
2 AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);  
3 DUMP employees;  
4 high_salary = FILTER employees BY salary > 60000;  
5 DUMP high_salary;
```

Run the script:

```
./pig -x mapreduce /home/bdalab/filter.pig
```

```
0 compute warning aggregation.  
2025-01-31 09:19:18,782 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2025-01-31 09:19:18,788 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.  
2025-01-31 09:19:18,815 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1  
2025-01-31 09:19:18,815 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(103,Bob,35,Finance,70000)  
2025-01-31 09:19:18,856 [main] INFO org.apache.pig.Main - Pig script completed in 5 minutes, 52 seconds and 222 milliseconds (352222 ms)  
bdalab@saraswati:/usr/local/pig/bin$
```

## Sorting Data: Sort employees by age in descending order.

Write Pig Latin Script and save it as sortd.pig

```
1 employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')  
2 AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);  
3 DUMP employees;  
4 sorted_employees = ORDER employees BY age DESC;  
5 DUMP sorted_employees;
```

Run the script:

```
./pig -x mapreduce /home/bdalab/sortd.pig
```

```
2025-01-31 09:33:33,256 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2025-01-31 09:33:33,264 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.  
2025-01-31 09:33:33,294 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1  
2025-01-31 09:33:33,294 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(104,Joy,36,Manager,55000)  
(103,Bob,35,Finance,70000)  
(101,John,30,HR,50000)  
(102,Alice,25,IT,60000)  
2025-01-31 09:33:33,336 [main] INFO org.apache.pig.Main - Pig script completed in 11 minutes, 35 seconds and 88 milliseconds (695088 ms)  
bdalab@saraswati:/usr/local/pig/bin$ 
```

## Aggregations: Average Salary by Department

Write Pig Latin Script and save it as average.pig

```
1 employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')  
2 AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);  
3 DUMP employees;  
4 grouped_data = GROUP employees BY department;  
5 avg_salary = FOREACH grouped_data GENERATE group AS department, AVG(employees.salary) AS  
    avg_salary;  
6 DUMP avg_salary;
```

Run the script:

```
./pig -x mapreduce /home/bdalab/average.pig
```

```
o compute warning aggregation.  
2025-01-31 09:47:39,527 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2025-01-31 09:47:39,529 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.  
2025-01-31 09:47:39,537 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1  
2025-01-31 09:47:39,537 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(HR,50000.0)  
(IT,60000.0)  
(Finance,70000.0)  
(Manager,55000.0)  
2025-01-31 09:47:39,605 [main] INFO org.apache.pig.Main - Pig script completed in 5 minutes, 50 seconds and 565 milliseconds (350565 ms)  
bdalab@saraswati:/usr/local/pig/bin$ 
```

## Joining Two Datasets

Create a .csv file (department.csv)

```
HR,Bangalore  
IT,Bangalore  
Finance,Mumbai  
Manager,NewYork
```

Load it on HDFS

```
hdfs dfs -put department.csv /user/studentregno/lab5
```

Write Pig Latin Script and save it as join.pig

```
1 employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')  
2 AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);  
3 departments = LOAD '/user/studentregno/lab5/department.csv' USING PigStorage(',')  
4 AS (dept_name:chararray, location:chararray);  
5 joined_data = JOIN employees BY department, departments BY dept_name;  
6 DUMP joined_data;
```

Run the script:

```
./pig -x mapreduce /home/bdalab/join.pig
```

```
2025-01-31 11:06:41,902 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!  
2025-01-31 11:06:41,914 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - yarn.resourcemanager.system-metrics-publisher.enabled  
is deprecated. Instead, use yarn.system-metrics-publisher.enabled  
2025-01-31 11:06:41,921 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.  
2025-01-31 11:06:41,957 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1  
2025-01-31 11:06:41,957 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1  
(101,John,30,HR,50000,HR,Bangalore)  
(102,Alice,25,IT,60000,IT,Bangalore)  
(103,Bob,35,Finance,70000,Finance,Mumbai)  
(104,Joy,36,Manager,55000,Manager,NewYork)  
2025-01-31 11:06:42,104 [main] INFO org.apache.pig.Main - Pig script completed in 3 minutes, 13 seconds and 833 milliseconds (193833 ms)  
bdalab@saraswati:/user/local/pig/bin$
```

## Store Processed Data to HDFS

Write Pig Latin Script and save it as storehdfs.pig

```
1 employees = LOAD '/user/studentregno/lab5/employee.csv' USING PigStorage(',')  
2 AS (emp_id:int, name:chararray, age:int, department:chararray, salary:int);  
3 DUMP employees;  
4 STORE employees INTO '/user/studentregno/lab5/output5/' USING PigStorage(',');  
5
```

Run the script:

```
./pig -x mapreduce /home/bdalab/storehdfs.pig
```

After completion can see the data in HDFS output directory:

```
hdfs dfs -cat /user/studentregno/lab5/output5/part-m-00000
```

#### **Week-5 Exercise (use Mapreduce mode):**

1. Create a sales dataset (order\_id,product,category,amount). Load it into Pig. Display the first 10 records
2. Load the sales dataset. Filter and display products where the amount is greater than 5000.
3. Load the sales dataset. Sort the products in descending order of amount. Display the sorted results.
4. Load the sales dataset. Group sales by category. Find the total sales amount for each category.
5. Create customer dataset (order\_id,customer\_name,city). Load sales dataset and customers datasets. Perform a JOIN operation on the order\_id column. Display the results.
6. Find the top 3 most expensive products from the sales dataset.
7. Store the filtered high-value transactions (amount > 5000) in an HDFS output directory.

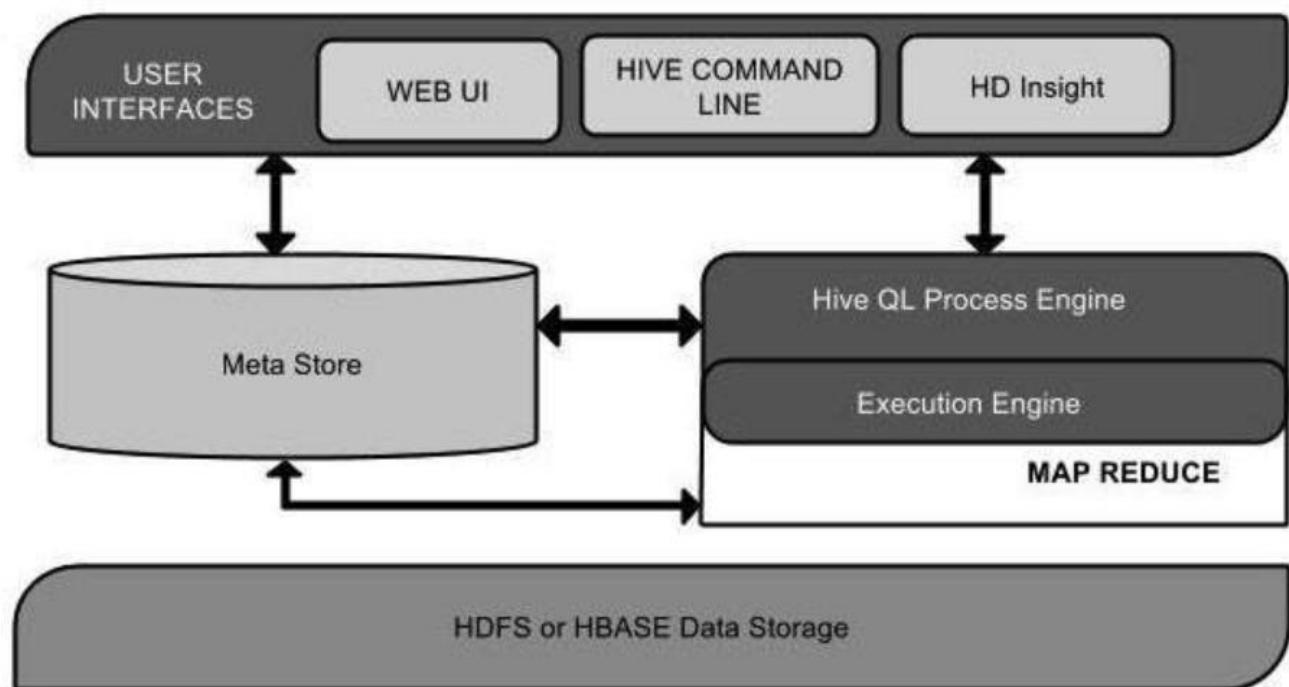
## Week-6 Exploring Hive

Apache Hive is a data warehouse infrastructure built on top of Apache Hadoop that enables SQL-like querying on large datasets stored in HDFS (Hadoop Distributed File System). It is primarily used for data analysis, ETL (Extract, Transform, Load) operations, and business intelligence.

### Why Use Hive?

- SQL-Like Queries – Uses HiveQL, which is similar to SQL.
- Handles Big Data – Efficiently processes petabytes of structured/unstructured data.
- Scalability – Runs on top of Hadoop, making it highly scalable.
- Schema-on-Read – Unlike traditional databases, data is interpreted at query time.
- Supports Various File Formats – Works with CSV, ORC, Parquet, JSON, and more.

### Hive Architecture



Hive consists of several key components:

1. **Metastore** – Stores metadata (schema, table definitions, partitions).
2. **Driver** – Manages query execution lifecycle.
3. **Query Engine** – Converts HiveQL into execution plans.

4. **Execution Engine** – Runs queries using MapReduce, Tez, or Spark.
5. **HDFS** – Stores the actual data.

## Key Features of Hive

- **Tables and Partitions** – Data is stored in tables that can be partitioned for faster queries.
- **User-Defined Functions (UDFs)** – Allows custom functions in Java/Python.
- **Joins and Aggregations** – Supports complex queries, joins, and analytics.
- **Optimization Features** – Partitioning, bucketing, indexing, and ORC file format.
- **Integration** – Works with Spark, Presto, Impala, and BI tools like Tableau.

## Hive vs. Traditional Databases

Feature	Hive	Traditional RDBMS
Query Language	HiveQL (SQL-like)	SQL
Data Processing	Batch (MapReduce, Tez, Spark)	OLTP (Real-time)
Schema	Schema-on-Read	Schema-on-Write
Speed	Slower for small queries	Fast
ACID Support	Limited (Only in newer versions)	Fully Supported
Scalability	High (Handles petabytes of data)	Moderate

## When to Use Hive?

- Processing large-scale data (e.g., logs, clickstream, IoT).
- Running batch analytics on Hadoop.
- ETL workloads and data preprocessing.
- Data exploration and reporting.
- Not ideal for: Real-time transactions, low-latency queries, or updates.

## **Working with Hive:**

### **Start Hadoop services**

```
start-dfs.sh
```

```
start-yarn.sh
```

Execute the following command to give the all permission to hive directory:

```
hdfs dfs -chmod 777 /tmp
```

### **Start Hive:**

```
cd /usr/local/hive
```

```
bin/schematool -dbType derby -initSchema
```

if any error then only run :

```
rm -rf metastore_db
```

If no error: then

```
bin/hive
```

```
bdalab@saraswati:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [saraswati]
bdalab@saraswati:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
bdalab@saraswati:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hive/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 8c45fd87-c91d-49b5-b1dc-b2933d47d1b8

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.3.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, t
ez) or using Hive 1.X releases.
hive> 
```

## **Hive Database Operations:**

### **Create a new database**

```
CREATE DATABASE sales_db;
```

### **List databases**

```
SHOW DATABASES;
```

### **Use a database**

```
USE sales_db;
```

Show Table list

```
SHOW TABLES;
```

### Drop a database

```
DROP DATABASE sales_db CASCADE;
```

```
hive> show databases;
OK
default
Time taken: 0.05 seconds, Fetched: 1 row(s)
hive> show tables;
OK
Time taken: 0.05 seconds
hive> 
Time taken: 0.05 seconds
hive> CREATE DATABASE sales_db;
OK
Time taken: 0.546 seconds
hive> SHOW DATABASES;
OK
default
sales_db
Time taken: 0.031 seconds, Fetched: 2 row(s)
hive> USE sales_db;
OK
Time taken: 0.024 seconds
hive> DROP DATABASE IF EXISTS test_db CASCADE;
OK
Time taken: 0.018 seconds
hive> 
```

### Solved Exercise:

E-Commerce Sales Analytics: An e-commerce company wants to analyze sales trends and customer purchases. The data is stored in Hadoop, and they need insights into total revenue, top customers, and monthly sales trends.

Sample data: (sales\_data.csv)

transaction_id	customer_id	product	category	amount	payment_type	transaction_date
1	101	Laptop	Electronics	1200	Credit Card	2023-01-05
2	102	Phone	Electronics	800	PayPal	2023-02-10
3	103	T-shirt	Clothing	50	Debit Card	2023-02-15
4	101	Headphones	Electronics	150	Credit Card	2023-03-01
5	104	Shoes	Footwear	120	Cash	2023-03-20

### Create sales\_data.csv (in any editor) and load it on HDFS

```

1 transaction_id, customer_id, product, category, amount, payment_type, transaction_date
2 1, 101, Laptop, Electronics, 1200, Credit Card, 2023-01-05
3 2, 102, Phone, Electronics, 800, PayPal, 2021-02-10
4 3, 103, T-shirt, Clothing, 50, Debit Card, 2022-02-15
5 4, 101, Headphones, Electronics, 150, Credit Card, 2023-03-01
6 5, 104, Shoes, Footwear, 120, Cash, 2022-03-20

```

```

hdfs dfs -mkdir /user/studentregno/lab6
hdfs dfs -put sales_data.csv /user/studentregno/lab6

```

#### a. Create a Hive database and table

```

CREATE DATABASE ecommerce;
USE ecommerce;

```

```

hive> CREATE DATABASE ecommerce;
OK
Time taken: 0.122 seconds
hive> USE ecommerce;
OK
Time taken: 0.059 seconds

```

```

CREATE TABLE sales ( transaction_id INT, customer_id INT, product STRING, category
STRING, amount FLOAT, payment_type STRING, transaction_date STRING ) ROW
FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE ;

```

#### To skip the header information:

```
alter table sales set tblproperties ("skip.header.line.count"="1");
```

```

hive> CREATE TABLE sales ( transaction_id INT, customer_id INT, product STRING, category STRING, amount FLOAT, payment_type STRING, transaction_date STRING ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE ;
OK
Time taken: 1.537 seconds
hive> alter table sales set tblproperties ("skip.header.line.count"="1");
OK
Time taken: 0.255 seconds
hive> DESCRIBE sales;
OK
transaction_id      int
customer_id         int
product             string
category            string
amount              float
payment_type        string
transaction_date    string
Time taken: 0.171 seconds, Fetched: 7 row(s)
hive> 

```

## b. Load Data from HDFS

LOAD DATA INPATH '/user/studentregno/lab6/sales\_data.csv' INTO TABLE sales;

```

hive> LOAD DATA INPATH '/user/studentregno/lab6/sales_data.csv' INTO TABLE sales;
Loading data to table ecommerce.sales
OK
Time taken: 1.246 seconds
hive> select *from sales;
OK
1      101      Laptop   Electronics     1200.0  Credit Card    2023-01-05
2      102      Phone    Electronics     800.0   PayPal       2021-02-10
3      103      T-shirt  Clothing      50.0    Debit Card    2022-02-15
4      101      Headphones Electronics    150.0   Credit Card    2023-03-01
5      104      Shoes    Footwear      120.0   Cash         2022-03-20
Time taken: 3.634 seconds, Fetched: 5 row(s)
hive> select *from sales where customer_id=101;
OK
1      101      Laptop   Electronics     1200.0  Credit Card    2023-01-05
4      101      Headphones Electronics    150.0   Credit Card    2023-03-01
Time taken: 1.15 seconds, Fetched: 2 row(s)
hive> 

```

## c. Analyze Total Revenue:

SELECT SUM(amount) AS total\_revenue FROM sales;

```
hive> SELECT SUM(amount) AS total_revenue FROM sales;
Query ID = bdalab_20250207094827_aca6febd-4bf6-43be-b795-68be37a18e2a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1738899093263_0001, Tracking URL = http://saraswati:8088/proxy/application_1738899093263_0001/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1738899093263_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-07 09:48:47,138 Stage-1 map = 0%, reduce = 0%
2025-02-07 09:48:56,736 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.89 sec
2025-02-07 09:49:06,357 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 6.32 sec
MapReduce Total cumulative CPU time: 6 seconds 320 msec
Ended Job = job_1738899093263_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 6.32 sec HDFS Read: 14392 HDFS Write: 106 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 320 msec
OK
2320.0
Time taken: 40.997 seconds, Fetched: 1 row(s)
hive> 
```

#### d. Identify the Most Popular Product

```
SELECT product, COUNT(*) AS sales_count
FROM sales
GROUP BY product
ORDER BY sales_count DESC
LIMIT 1;
```

```

Time taken: 16.952 seconds, Fetched: 1 row(s)
hive> SELECT product, COUNT(*) AS sales_count FROM sales GROUP BY product ORDER BY sales_count DESC LIMIT 1;
Query ID = bdalab_20250207095238_488c25c7-4aac-4607-a278-731200aecc1d
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1738899093263_0002, Tracking URL = http://sarawati:8088/proxy/application_1738899093263_0002/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1738899093263_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-07 09:52:50,704 Stage-1 map = 0%, reduce = 0%
2025-02-07 09:53:02,398 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.79 sec
2025-02-07 09:53:15,001 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 5.86 sec
MapReduce Total cumulative CPU time: 5 seconds 860 msec
Ended Job = job_1738899093263_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1738899093263_0003, Tracking URL = http://sarawati:8088/proxy/application_1738899093263_0003/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1738899093263_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-02-07 09:53:31,787 Stage-2 map = 0%, reduce = 0%
2025-02-07 09:53:37,052 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 2.42 sec
2025-02-07 09:53:43,387 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 5.27 sec
MapReduce Total cumulative CPU time: 5 seconds 270 msec
Ended Job = job_1738899093263_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.86 sec HDFS Read: 13801 HDFS Write: 224 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 5.27 sec HDFS Read: 7817 HDFS Write: 112 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 130 msec
OK
Headphones      1
Time taken: 66.952 seconds, Fetched: 1 row(s)
hive> 
```

### e. Analyze Sales by Payment Method

```

SELECT payment_type, COUNT(*) AS transaction_count
FROM sales
GROUP BY payment_type;
```

```

Time taken: 35.547 seconds, Fetched: 4 row(s)
hive> SELECT payment_type, COUNT(*) AS transaction_count FROM sales GROUP BY payment_type;
Query ID = bdalab_20250207095915_de4ffff6f-5589-4bf4-bcd9-d544983ba160
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1738899093263_0004, Tracking URL = http://sarawati:8088/proxy/application_1738899093263_0004/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1738899093263_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-07 09:59:25,737 Stage-1 map = 0%, reduce = 0%
2025-02-07 09:59:37,833 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 8.88 sec
2025-02-07 09:59:46,173 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 11.72 sec
MapReduce Total cumulative CPU time: 11 seconds 720 msec
Ended Job = job_1738899093263_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 11.72 sec HDFS Read: 14683 HDFS Write: 178 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 720 msec
OK
Cash      1
Credit Card    2
Debit Card     1
PayPal      1
Time taken: 35.547 seconds, Fetched: 4 row(s)
hive> 
```

## f. Optimizing the Table with Partitioning:

For large datasets, partitioning helps speed up queries.

### Enable Dynamic Partitioning:

```
SET hive.exec.dynamic.partition = true;
```

```
SET hive.exec.dynamic.partition.mode = nonstrict;
```

### Create partition table (sales table by year):

```
CREATE TABLE sales_partitioned (transaction_id INT, customer_id INT, product STRING,
category STRING, amount FLOAT, payment_type STRING) PARTITIONED BY (year INT)
STORED AS TEXTFILE;
```

### Load Data into Partitions:

```
INSERT INTO sales_partitioned PARTITION (year) SELECT transaction_id, customer_id,
product, category, amount, payment_type, YEAR(TO_DATE(transaction_date)) AS year FROM
sales;
```

```
hive> SET hive.exec.dynamic.partition = true;
hive> SET hive.exec.dynamic.partition.mode = nonstrict;
hive> INSERT INTO sales_partitioned PARTITION (year) SELECT transaction_id, customer_id, product, category, amount, payment_type, YEAR(TO_DATE
(transaction_date)) AS year FROM sales;
Query ID = bdalab_20250214090200_9d71c0bd-2fb1-4242-82a8-f5e1ab977db4
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1739501325477_0002, Tracking URL = http://sarawati:8088/proxy/application_1739501325477_0002/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1739501325477_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-02-14 09:02:11,340 Stage-1 map = 0%, reduce = 0%
2025-02-14 09:02:22,845 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.71 sec
2025-02-14 09:02:31,172 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 8.58 sec
MapReduce Total cumulative CPU time: 8 seconds 580 msec
Ended Job = job_1739501325477_0002
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/ecommerce.db/sales_partitioned/.hive-staging_hive_2025-02-14_09-02-00_504_5
11226748826042543-1/-ext-10000
Loading data to table ecommerce.sales_partitioned partition (year=null)

      Time taken to load dynamic partitions: 0.787 seconds
      Time taken for adding to write entity : 0.003 seconds
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1  Cumulative CPU: 8.58 sec  HDFS Read: 22643 HDFS Write: 1293 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 580 msec
OK
Time taken: 35.241 seconds
hive> []
```

```
SHOW PARTITIONS sales_partitioned;
```

### Query the partitioned Table:

```
SELECT * FROM sales_partitioned WHERE year = 2023;
```

```
hive> SHOW PARTITIONS sales_partitioned;
OK
year=2021
year=2022
year=2023
Time taken: 0.165 seconds, Fetched: 3 row(s)
hive> SELECT * FROM sales_partitioned WHERE year = 2023;
OK
1      101    Laptop   Electronics     1200.0  Credit Card    2023
4      101    Headphones   Electronics     150.0   Credit Card    2023
Time taken: 0.725 seconds, Fetched: 2 row(s)
```

### g. bucketed table:

Bucketing in Hive is used to **divide data into fixed-size buckets** based on a column's hash value. This improves **query performance** by reducing the number of files scanned.

#### Create a bucketed table:

- **CLUSTERED BY (customer\_id)** → Used for bucketing
- **INTO 4 BUCKETS** → Data will be divided into 4 buckets

```
CREATE TABLE retail_sales_bucketed (
```

```
    transaction_id INT,
    customer_id INT,
    product STRING,
    category STRING,
    amount FLOAT,
    payment_type STRING
    transaction_date STRING
)
```

```
CLUSTERED BY (customer_id) INTO 4 BUCKETS STORED AS TEXTFILE;
```

```
Time taken: 0.161 seconds
hive> CREATE TABLE sales_bucketed (transaction_id INT, customer_id INT, product STRING, category STRING, amount FLOAT, payment_type STRING,transaction_date STRING) CLUSTERED BY (customer_id) INTO 4 BUCKETS STORED AS TEXTFILE;
OK
Time taken: 0.184 seconds
hive> show tables;
OK
retail_sales_partitioned
sales
sales_bucketed
sales_partitioned
Time taken: 0.05 seconds, Fetched: 4 row(s)
hive> 
```

## Enable Bucketing in Hive:

```
SET hive.enforce.bucketing = true;
```

## Load Data into Bucketed Table:

```
INSERT OVERWRITE TABLE sales_bucketed SELECT * FROM sales SORT BY
customer_id;
```

```
hive> SET hive.enforce.bucketing = true;
hive> INSERT OVERWRITE TABLE sales_bucketed SELECT * FROM sales SORT BY customer_id;
Query ID = bdalab_20250214093659_20f84efd-6aed-488a-8da3-8307ed7ba294
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks determined at compile time: 4
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1739501325477_0003, Tracking URL = http://sarawati:8088/proxy/application_1739501325477_0003/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1739501325477_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 4
2025-02-14 09:37:13,720 Stage-1 map = 0%, reduce = 0%
2025-02-14 09:37:25,203 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.75 sec
2025-02-14 09:37:43,326 Stage-1 map = 100%, reduce = 25%, Cumulative CPU 7.68 sec
2025-02-14 09:37:46,524 Stage-1 map = 100%, reduce = 50%, Cumulative CPU 12.47 sec
2025-02-14 09:37:48,635 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 22.39 sec
MapReduce Total cumulative CPU time: 22 seconds 390 msec
Ended Job = job_1739501325477_0003
Loading data to table ecommerce.sales_bucketed
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1739501325477_0004, Tracking URL = http://sarawati:8088/proxy/application_1739501325477_0004/
Kill Command = /usr/local/hadoop/bin/mapred job -kill job_1739501325477_0004
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 1
2025-02-14 09:38:33,280 Stage-3 map = 0%, reduce = 0%
2025-02-14 09:38:40,596 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 2.68 sec
2025-02-14 09:38:48,951 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 5.95 sec
MapReduce Total cumulative CPU time: 5 seconds 950 msec
Ended Job = job_1739501325477_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 4 Cumulative CPU: 22.39 sec HDFS Read: 49645 HDFS Write: 1854 SUCCESS
Stage-Stage-3: Map: 1 Reduce: 1 Cumulative CPU: 5.95 sec HDFS Read: 18098 HDFS Write: 490 SUCCESS
Total MapReduce CPU Time Spent: 28 seconds 340 msec
OK
Time taken: 113.366 seconds
hive> █
```

## Querying Data Using Buckets:

```
SELECT * FROM sales_bucketed  
TABLESAMPLE(BUCKET 2 OUT OF 4 ON customer_id);  
(BUCKET 2 OUT OF 4 → Fetches data only from bucket 2)
```

```
hive> SELECT * FROM sales_bucketed TABLESAMPLE(BUCKET 2 OUT OF 4 ON customer_id);  
OK  
2      102     Phone   Electronics    800.0   PayPal   2021-02-10  
3      103     T-shirt Clothing     50.0    Debit Card   2022-02-15  
Time taken: 0.252 seconds, Fetched: 2 row(s)  
hive> █
```

## Week-6 Exercises:

1. A tech company wants to analyze web server logs to monitor website traffic, detect peak usage times, and track 404 errors. Do the Log analysis of server for the dataset (weblogs.csv)

log_id	ip_address	url	status_code	response_time	log_date
1	192.168.1.1	/home	200	120ms	2024-01-01
2	192.168.1.2	/product	404	60ms	2024-01-01
3	192.168.1.3	/checkout	500	200ms	2024-01-02

- a) Create a Hive table for log data. Load Data from HDFS  
b) Find the Most Visited URLs  
c) Detect Peak Traffic Hours  
d) Find 404 Errors  
e) Create a partitioned Hive table based on date. How would you insert data dynamically into partitions?  
f) Create a Bucketed Table by status code. Load data into the bucketed table. Retrieve 404 errors (Bucketed Table)
2. Analyze movie ratings, identify top movies, and understand user behavior using **IMDb-style data**.

Dataset: movie\_ratings.csv

```

movie_id,title,genre,release_year,user_id,rating,timestamp
101,The Dark Knight,Action,2008,5001,9.0,2024-01-01 12:30:00
102,Inception,Sci-Fi,2010,5002,8.8,2024-01-01 14:00:00
103,Parasite,Thriller,2019,5003,8.6,2024-01-02 16:45:00
104,Interstellar,Sci-Fi,2014,5004,8.6,2024-01-03 10:30:00
105,Avengers: Endgame,Action,2019,5005,8.4,2024-01-04 09:00:00
106,Spirited Away,Animation,2001,5006,8.6,2024-01-05 22:15:00

```

- a) Create a Hive table for movie. Load Data from HDFS
  - b) Find the Top 5 Highest Rated Movies
  - c) Find the Most Popular Genre
  - d) Extract Yearly Trends (Extract Year from Review Date)
  - e) Create a Partitioned Table by Genre. Load Data into Partitions. Retrieve All Sci-Fi Movies (from Partitioned Table)
  - f) Create a Bucketed Table by release\_year. Load Data. Retrieve Ratings of Movies Released in 2019 (Bucketed Table)
3. Analyze flight delays, on-time performance, and airport efficiency using real-world airline data.

Dataset: flight\_data.csv

```

flight_id,airline,flight_number,origin,destination,departure_date,departure_time,arrival_time,delay_minutes,status
101,Delta,DL202,JFK,LAX,2024-02-01,08:30,11:30,10,On Time
102,United,UA405,ORD,SFO,2024-02-01,09:00,12:00,45,Delayed
103,American,AA100,LAX,MIA,2024-02-02,10:00,14:00,0,On Time
104,Southwest,SW305,DFW,DEN,2024-02-02,11:30,13:30,30,Delayed
105,JetBlue,JB678,BOS,ATL,2024-02-03,07:00,09:30,5,On Time

```

- a) Create a Hive table for flight data. Load Data from HDFS
- b) Find the Airline with the Most Delays
- c) Count the Number of Flights by Status (On Time, Delayed)
- d) Calculate the Average Delay per Airline
- e) Create a Partitioned Table by Departure Date. Load Data into Partitions.
- f) Create a Bucketed Table by Airline. Load data into the bucketed table. Retrieve Flights Operated by Delta (Bucketed Table)

## Week-7 Explore HBase (basic commands)

HBase is a distributed, scalable, and NoSQL database that runs on top of Hadoop HDFS. It is modeled after Google Bigtable and designed for real-time, read/write access to large datasets.

### Key Features:

- **Schema-less:** Only column families are predefined. Columns can be added dynamically.
- **Auto-Scalability:** Scales horizontally across multiple nodes.
- **High Write Throughput:** Supports fast inserts and updates.
- **Strong Consistency:** Guarantees consistent reads and writes per row.
- **Fault Tolerance:** Uses HDFS for durability and replication.

### HBase vs. RDBMS

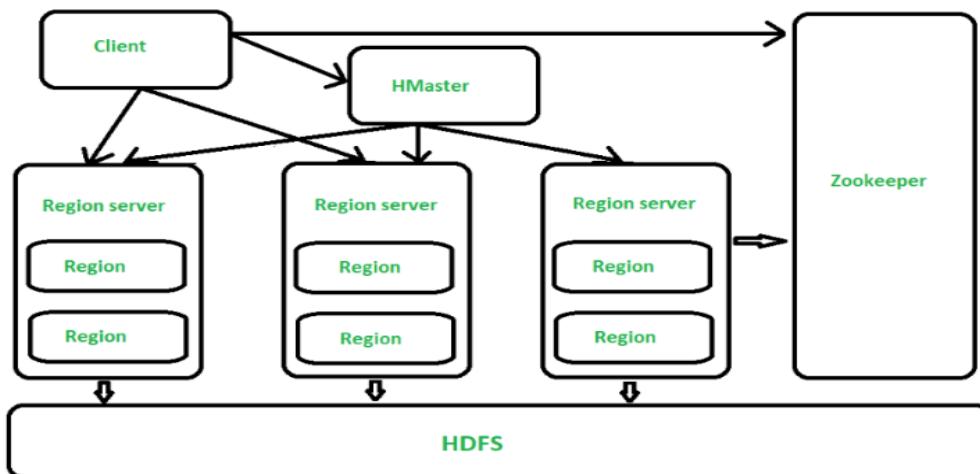
Feature	HBase (NoSQL)	RDBMS (SQL)
Data Model	Column-family-based	Table-based (rows/columns)
Schema	Dynamic (No predefined columns)	Fixed schema
Scalability	Horizontal (adds nodes)	Vertical (upgrading hardware)
Transactions	Row-level atomicity	ACID transactions
Query Language	NoSQL (HBase API, Phoenix SQL)	SQL
Use Case	Large-scale analytics, big data	Traditional business applications

### HBase Architecture

HBase follows a **master-slave** architecture consisting of several key components:

### Main Components:

- **HMaster:** Coordinates regions, manages schema, and load balancing.
- **RegionServers:** Handle read/write requests, manage HBase tables.
- **HDFS (Hadoop Distributed File System):** Stores the actual data in blocks.
- **ZooKeeper:** Maintains metadata and handles failover.



## Data Storage Model in HBase

HBase organizes data into **tables**, which contain:

- **Rows** (identified by unique **row keys**)
- **Column Families** (predefined storage units)
- **Columns** (dynamic)
- **Timestamps** (versions of data)

## Basic HBase Commands (HBase Shell):

### Start Hadoop services:

`start-dfs.sh`

`start-yarn.sh`

### Next Start HBase with following commands

`cd /usr/local/hbase`

`bin/start-hbase.sh`

### Now check status again by

`jps`

(you should able to see `HRegionServer`, `HQuorumPeer`, `HMaster`)

```

bdalab@saraswati:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [saraswati]
bdalab@saraswati:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
bdalab@saraswati:~$ cd /usr/local/hbase/
bdalab@saraswati:/usr/local/hbase$ bin/start-hbase.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/client-facing-thirdparty/log4j-slf4j-impl-2.17.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/client-facing-thirdparty/log4j-slf4j-impl-2.17.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
127.0.0.1: running zookeeper, logging to /usr/local/hbase/bin/../logs/hbase-bdalab-zookeeper-saraswati.out
running master, logging to /usr/local/hbase/logs/hbase-bdalab-master-saraswati.out
: running regionserver, logging to /usr/local/hbase/logs/hbase-bdalab-regionserver-saraswati.out
bdalab@saraswati:/usr/local/hbase$ jps
8400 Jps
8225 HRegionServer
6261 NameNode
6598 SecondaryNameNode
7014 NodeManager
6407 DataNode
6874 ResourceManager
8043 HMaster
7932 HQuorumPeer
bdalab@saraswati:/usr/local/hbase$ 
```

## Next run hbase shell:

./bin/hbase shell

```

Connected(1453)) - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x1000016acfb0002, negotiated timeout = 900
00
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.5.10, ra3af60980c61fb4be31e0dcdb89880f304d01098a, Thu Jul 18 22:45:17 PDT 2024
Took 0.0019 seconds
hbase:001:0> 
```

```

hbase:001:0> list
TABLE
0 row(s)
Took 0.4829 seconds
=> []
hbase:002:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.1441 seconds
hbase:003:0> 2025-02-26 00:16:19,015 INFO  [ReadOnlyZKClient-127.0.0.1:2181@0x15e8c040] zookeeper.ZooKeeper (ZooKeeper.java:close(1232)) - Session: 0x1000007fecb0002 closed
2025-02-26 00:16:19,015 INFO  [ReadOnlyZKClient-127.0.0.1:2181@0x15e8c040-EventThread] zookeeper.ClientCnxn (ClientCnxn.java:run(569)) - Event Thread shut down for session: 0x1000007fecb0002 
```

## Creating a Table

create 'students', 'info', 'contact'

## Inserting Data

put 'students', '1', 'info:name', 'Alice'

put 'students', '1', 'info:age', '22'

```
put 'students', '1', 'contact:email', 'alice@gmail.com'
```

## Retrieving Data

```
get 'students', '1'  
scan 'students'
```

```
hbase:009:0> get 'students', '1'  
COLUMN CELL  
 contact:email timestamp=2025-02-26T00:18:27.331, value=alice@gmail.com  
 info:age timestamp=2025-02-26T00:17:59.845, value=22  
 info:name timestamp=2025-02-26T00:17:07.882, value=Alice  
1 row(s)  
Took 0.1152 seconds  
hbase:010:0> scan 'students'  
ROW COLUMN+CELL  
 1 column=contact:email, timestamp=2025-02-26T00:18:27.331, value=alice@gmail.com  
 1 column=info:age, timestamp=2025-02-26T00:17:59.845, value=22  
 1 column=info:name, timestamp=2025-02-26T00:17:07.882, value=Alice  
1 row(s)  
Took 0.0223 seconds  
hbase:011:0> □
```

## Updating Data

```
put 'students', '1', 'contact:email', 'alice_new@gmail.com'
```

## Deleting Data

```
delete 'students', '1', 'contact:email'  
deleteall 'students', '1'
```

## Disabling a Table

```
disable 'tablename'  
is_disabled 'table name'
```

## Enabling a Table

```
enable 'tablename'  
is_enabled 'table name'
```

## describe

```
describe 'table name'
```

## Dropping a Table

```
disable 'students'
```

```
drop 'students'
```

```
hbase:011:0> put 'students', '1', 'contact:email', 'alice_new@gmail.com'
Took 0.0246 seconds
hbase:012:0> delete 'students', '1', 'contact:email'
Took 0.2185 seconds
hbase:013:0> deleteall 'students', '1'
Took 0.0092 seconds
hbase:014:0> disable 'students'
2025-02-26 00:28:29,982 INFO [main] client.HBaseAdmin (HBaseAdmin.java:rpcCall(927)) - Started disable of students
2025-02-26 00:28:32,238 INFO [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3599)) - Operation: DISABLE, Table Name: default:students, procId: 12 completed
Took 2.5264 seconds
hbase:015:0> drop 'students'
2025-02-26 00:28:49,261 INFO [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3599)) - Operation: DELETE, Table Name: default:students, procId: 15 completed
Took 1.2793 seconds
hbase:016:0> □
```

## Common Filters in HBase

### SingleColumnValueFilter: Filter rows based on a column's value.

```
scan 'students', {FILTER => "SingleColumnValueFilter ('info', 'age', >=, 'binary:20')"}
```

```
hbase:026:0> scan 'students'
ROW
1
1
1
2
2
2
2 row(s)
Took 0.0129 seconds
hbase:027:0> scan 'students', {FILTER => "SingleColumnValueFilter ('info', 'age', >=, 'binary:21')"}
ROW
1
1
1
1 row(s)
Took 0.0579 seconds
hbase:028:0> □
```

### PrefixFilter: Fetch records where the row key starts with a given prefix.

```
scan 'students', {FILTER => "PrefixFilter('2')"}  
ROW
```

```
hbase:031:0> scan 'students', {FILTER => "PrefixFilter('2')"}  
ROW  
 2  
 2  
 2  
1 row(s)  
Took 0.0126 seconds  
hbase:032:0> □
```

## **PageFilter: Implement pagination.**

```
scan 'students', {FILTER => "PageFilter(1)"}  
;
```

```
Took 0.0120 seconds
hbase:032:0> scan 'students', {FILTER => "PageFilter(1)"}
ROW                                COLUMN+CELL
 1                                  column=contact:email, timestamp=2025-02-26T00:34:35.036, value=alice@gmail.com
 1                                  column=info:age, timestamp=2025-02-26T00:34:25.551, value=22
 1                                  column=info:name, timestamp=2025-02-26T00:34:15.110, value=Alice
1 row(s)
Took 0.0790 seconds
hbase:033:0> scan 'students', {FILTER => "PageFilter(2)"}
ROW                                COLUMN+CELL
 1                                  column=contact:email, timestamp=2025-02-26T00:34:35.036, value=alice@gmail.com
 1                                  column=info:age, timestamp=2025-02-26T00:34:25.551, value=22
 1                                  column=info:name, timestamp=2025-02-26T00:34:15.110, value=Alice
 2                                  column=contact:email, timestamp=2025-02-26T00:35:54.410, value=abc@gmail.com
 2                                  column=info:age, timestamp=2025-02-26T00:35:36.188, value=20
 2                                  column=info:name, timestamp=2025-02-26T00:35:21.982, value=Abc
2 row(s)
Took 0.0153 seconds
hbase:034:0> □
```

```
hbase:005:0> create 'students', 'info'
2025-02-26 11:13:51,415 INFO [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3599)) - Operation: CREATE, Table Name: default:students, procId: 42 completed
Created table students
Took 3.3516 seconds
=> Hbase::Table - students
hbase:006:0> put 'students', '1', 'info:name', 'Alice'
Took 0.3654 seconds
hbase:007:0> put 'students', '1', 'info:age', '20'
Took 0.1870 seconds
hbase:008:0>
hbase:009:0> put 'students', '2', 'info:name', 'Bob'
Took 0.0076 seconds
hbase:010:0> put 'students', '2', 'info:age', '21'
Took 0.0063 seconds
hbase:011:0>
hbase:012:0> put 'students', '3', 'info:name', 'Charlie'
Took 0.0043 seconds
hbase:013:0> put 'students', '3', 'info:age', '20'
Took 0.0047 seconds
hbase:014:0>
hbase:015:0> put 'students', '4', 'info:name', 'David'
Took 0.0045 seconds
hbase:016:0> put 'students', '4', 'info:age', '22'
Took 0.0082 seconds
hbase:017:0>
hbase:018:0> put 'students', '5', 'info:name', 'Eve'
Took 0.0044 seconds
hbase:019:0> put 'students', '5', 'info:age', '21'
Took 0.0128 seconds
hbase:020:0> scan 'students'
ROW                                     COLUMN+CELL
1                                         column=info:age, timestamp=2025-02-26T11:14:06.018, value=20
1                                         column=info:name, timestamp=2025-02-26T11:14:05.984, value=Alice
2                                         column=info:age, timestamp=2025-02-26T11:14:06.371, value=21
2                                         column=info:name, timestamp=2025-02-26T11:14:06.340, value=Bob
3                                         column=info:age, timestamp=2025-02-26T11:14:06.432, value=20
3                                         column=info:name, timestamp=2025-02-26T11:14:06.402, value=Charlie
```

## **Week-7 Exercises:**

1.

- a) Create a table students with column families: "personal" (stores name, age), "academic" (stores marks, grade)

Create a table employees with column families: "details" (stores name, age, department) "salary" (stores basic pay, bonus)

Create a table orders with: Row Key as OrderID Column families: "customer" (name, email), "items" (product, quantity), "payment" (mode, amount)

- b) Describe the structure of all tables to verify creation.
- c) Delete the table orders and recreate it.
- d) Insert data into each table
- e) Retrieve only the Name and Marks of all students.
- f) Find all employees whose basic pay is greater than 55000.
- g) Display only the top 3 orders sorted by OrderID.
- h) Find all students who scored more than 80 marks using filtering
- i) Find all employees who belong to the IT department.
- j) Retrieve all orders where payment mode is "Credit Card".
- k) Count the total number of rows in employees.
- l) Display the oldest student using a scan filter.

2.

- a) Create a table library with column families: "book\_info" → (title, author, genre), "availability" → (status, borrower)
- b) Insert data into each table
- c) Retrieve all details of book B002.
- d) Find all books that are "Available".
- e) Update the status of book B003 to "Borrowed" and set borrower as "Michael".
- f) Count the number of books in the library.

3.

- a) Create a table transactions with column families: "customer\_info" → (name, email), "transaction\_details" → (amount, mode, date)
- b) Insert data into the table

- c) Retrieve all transactions where the amount is greater than 3000.
- d) Find all transactions made using "Credit Card".
- e) Retrieve only customer names and their transaction amounts.
- f) Delete transaction T002 from the table.

## **Week-9 Apache Spark**

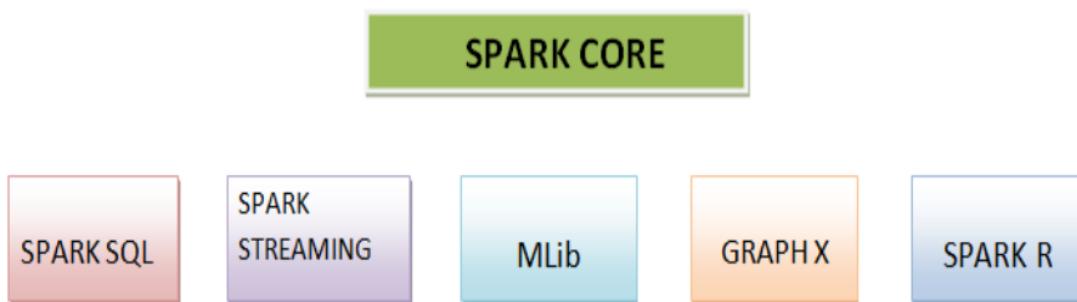
Apache Spark is an open-source, distributed computing system for fast and efficient big data processing. It provides an easy-to-use interface for processing large-scale datasets across clusters of computers. Spark extends the MapReduce model but computes in memory, making it 100x faster than Hadoop for specific workloads.

### **Key Features of Apache Spark**

- Speed – Performs in-memory computation for faster execution.
- Ease of Use – Supports Python, Scala, Java, and SQL with a simple API.
- Flexibility – Works with batch, streaming, and interactive data.
- Distributed Computing – Processes data across multiple machines for scalability.
- Integration – Works well with HDFS, HBase, Cassandra, Kafka, and more.

### **Apache Spark Components**

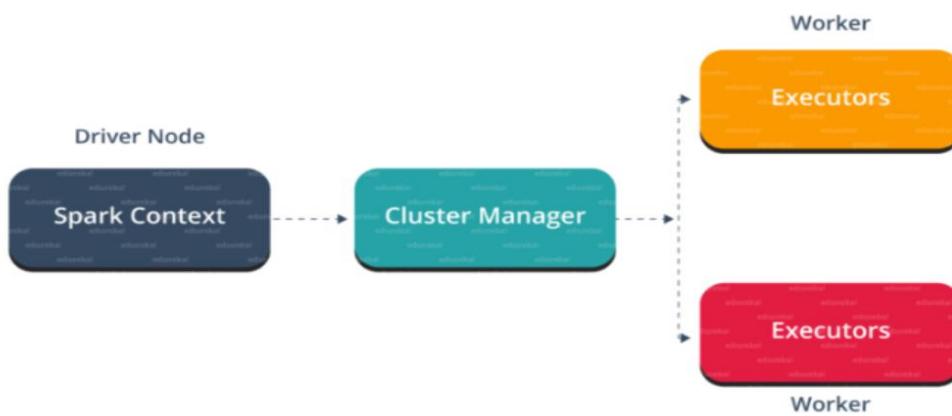
1. Spark Core – The foundation for all Spark applications, handling scheduling and memory management.
2. Spark SQL – Supports structured data processing using SQL queries and DataFrames.
3. Spark Streaming – Enables real-time data processing from sources like Kafka and Flume.
4. MLlib (Machine Learning Library) – Provides machine learning algorithms for classification, clustering, and recommendations.
5. GraphX – Used for graph processing and analytics.



## Apache Spark vs Hadoop

Feature	Apache Spark	Hadoop MapReduce
Speed	Very fast (in-memory)	Slower (disk-based)
Ease of Use	Simple APIs in Python, Scala, Java	Complex Java-based coding
Data Processing	Batch & Streaming	Batch only
Integration	Supports HDFS, HBase, Kafka, etc.	Mainly HDFS
Fault Tolerance	Automatic recovery using RDDs	Checkpoint-based recovery

## Spark Architecture :



## What is an RDD?

Resilient Distributed Dataset (RDD) is the fundamental data structure in Apache Spark. It is an immutable, distributed collection of objects that can be processed in parallel across a cluster.

## Key Features of RDDs:

- Resilient – Automatically recovers from node failures.
- Distributed – Data is split across multiple nodes for parallel processing.
- Immutable – Once created, an RDD cannot be modified, but transformations create new RDDs.
- Lazy Evaluation – Computations are delayed until an action is triggered.
- Fault Tolerance – Uses lineage (a history of transformations) to recompute lost data.

## Working with Spark:

Start PySpark Shell (Type the following command in Terminal)

```
pyspark
```

```
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.vectorized.if.expr.mode does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.exim.test.mode does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.query.results.cache.directory does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.query.results.cache.wait.for.pending.results does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.remove.orderby.in.subquery does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.tez.bmj.use.subcache does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.llap.io.vrb.queue.limit.min does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.server2.wm.pool.metrics does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.repl.add.raw.reserved.namespace does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.resource.use.hdfs.location does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.stats.num.nulls.estimate.percent does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.llap.io.acid does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.llap.zk.sm.session.timeout does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.vectorized.ptf.max.memory.buffering.batch.count does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.llap.task.scheduler.am.registry does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.druid.overlord.address.default does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.optimize.remove.sq_count_check does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.server2.webui.enable.cors does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.vectorized.row.serde.inputformat.excludes does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.query.reexecution.stats.cache.size does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.combine.equivalent.work.optimization does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.lock.query.string.max.length does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.llap.io.track.cache.usage does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.use.orc.codec.pool does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.query.results.cache.max.size does not exist
25/02/26 12:16:41 WARN HiveConf: HiveConf of name hive.repl.bootstrap.dump.open.txn.timeout does not exist
Welcome to
   ____/
  / \ \_ \ \_ \ \_ \ \_ \ \_ \
 / \ / . \ \_, \_ / \_ / \_ / \_ \
  \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \
                                             version 3.1.3
Using Python version 3.8.10 (default, Feb  4 2025 15:02:54)
Spark context Web UI available at http://sarasmarti.mshome.net:4040
Spark context available as 'sc' (master = local[*], app id = local-1740552404873).
SparkSession available as 'spark'.
>>> 
```

## Create an RDD (Resilient Distributed Dataset)

There are three ways to create an RDD in Spark:

From an existing collection (parallelized)

From an external data source (e.g., HDFS, S3, CSV, JSON, etc.)

By transforming existing RDDs

```
# Create an RDD from a list
rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])
print(rdd.collect())
```

## Perform a Simple Transformation

```
# Multiply each element by 2  
rdd_transformed = rdd.map(lambda x: x * 2)  
print(rdd_transformed.collect())
```

## **Creating a DataFrame from a CSV file:**

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName("DataFrame Example").getOrCreate()
df = spark.read.csv("sample_data.csv", header=True, inferSchema=True)
df.show()
```

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.appName("DataFrame Example").getOrCreate()
>>> df = spark.read.csv("sample_data.csv", header=True, inferSchema=True)
>>> df.show()
+---+-----+-----+
| id| name|age| dept|salary|
+---+-----+-----+
| 101| John| 30| HR| 50000|
| 102| Alice| 25| IT| 60000|
| 103| Bob| 35| Finance| 70000|
| 104| Joy| 36| Manager| 55000|
+---+-----+-----+
>>> □
```

### Selecting and Filtering Data:

```
# Select specific columns
df.select("name", "age").show()

# Filter records where age > 30
df.filter(df.age > 30).show()
```

```
>>> df.select("name", "age").show()
+---+-----+
| name|age|
+---+-----+
| John| 30|
| Alice| 25|
| Bob| 35|
| Joy| 36|
+---+-----+

>>> df.filter(df.age > 30).show()
+---+-----+-----+
| id|name|age| dept|salary|
+---+-----+-----+
| 103| Bob| 35| Finance| 70000|
| 104| Joy| 36| Manager| 55000|
+---+-----+
```

### Saving a DataFrame to a CSV File:

```
df.write.csv("output_data.csv", header=True)
```

## Running Spark Applications

Save the Python script as spark\_app.py and run (from path /usr/local/spark/bin):

```
spark-submit spark_app.py
```

### Sample Exercise:

Word Count program using Spark RDDs in Python

Steps :

1. Create input file sample.txt and load it on hdfs
2. Load a text file into an RDD.
3. Split each line into words using flatMap().
4. Create key-value pairs (word -> 1) using map().
5. Use reduceByKey() to count occurrences of each word.
6. Print the results.

1. Write Python Code (spark\_word\_count.py)

```
1 from pyspark import SparkContext
2
3 # Initialize SparkContext
4 sc = SparkContext("local", "Word Count HDFS")
5
6 # Define HDFS input and output paths
7 hdfs_input_path = "hdfs://localhost:9000/user/studentregno/lab9/sample.txt"
8 hdfs_output_path = "hdfs://localhost:9000/user/studentregno/lab9/output1/"
9
10 # Load text file from HDFS into an RDD
11 rdd = sc.textFile(hdfs_input_path)
12
13 # Perform word count
14 word_counts = (rdd.flatMap(lambda line: line.split()) # Split lines into words
15                 .map(lambda word: (word, 1))           # Map each word to (word, 1)
16                 .reduceByKey(lambda a, b: a + b))      # Reduce by key to count occurrences
17
18 # Save the result back to HDFS
19 word_counts.saveAsTextFile(hdfs_output_path)
20
21 # Stop SparkContext
22 sc.stop()
```

```

bdalab@saraswati:~$ hdfs dfs -mkdir /user/studentregno/lab9
bdalab@saraswati:~$ hdfs dfs -put sample.txt /user/studentregno/lab9
bdalab@saraswati:~$ hdfs dfs -mkdir /user/studentregno/lab9/output
bdalab@saraswati:~$ 

```

## 2. Execute it using spark-submit:

```

spark-submit spark_wordcount.py
bdalab@saraswati:/usr/local/spark/bin$ spark-submit /home/bdalab/spark_wordcount.py
25/03/11 00:46:11 WARN Utils: Your hostname, saraswati resolves to a loopback address: 127.0.1.1; using 192.168.29.104 instead (on interface wlp2s0)
25/03/11 00:46:11 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
25/03/11 00:46:11 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
25/03/11 00:46:12 INFO SparkContext: Running Spark version 3.1.3
25/03/11 00:46:12 INFO ResourceUtils: ======
25/03/11 00:46:12 INFO ResourceUtils: No custom resources configured for spark.driver.
25/03/11 00:46:12 INFO ResourceUtils: ======
25/03/11 00:46:12 INFO SparkContext: Submitted application: Word Count HDFS
25/03/11 00:46:12 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpu -> name: cpus, amount: 1.0)
25/03/11 00:46:12 INFO ResourceProfile: Limiting resource is cpu
25/03/11 00:46:12 INFO ResourceProfileManager: Added ResourceProfile id: 0
25/03/11 00:46:12 INFO SecurityManager: Changing view acls to: bdalab
25/03/11 00:46:12 INFO SecurityManager: Changing modify acls to: bdalab
25/03/11 00:46:12 INFO SecurityManager: Changing view acls groups to:
25/03/11 00:46:12 INFO SecurityManager: Changing modify acls groups to:
25/03/11 00:46:12 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(bdalab); groups with view permissions: Set(); users with modify permissions: Set(bdalab); groups with modify permissions: Set()
25/03/11 00:46:12 INFO Utils: Successfully started service 'sparkDriver' on port 34223.
25/03/11 00:46:12 INFO SparkEnv: Registering MapOutputTracker
25/03/11 00:46:12 INFO SparkEnv: Registering BlockManagerMaster
25/03/11 00:46:12 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
25/03/11 00:46:12 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
25/03/11 00:46:12 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
25/03/11 00:46:12 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-a014e2b6-08ca-484b-aeb8-a9eb1b88af57
25/03/11 00:46:12 INFO MemoryStore: MemoryStore started with capacity 306.3 MiB
25/03/11 00:46:12 INFO SparkEnv: Registering OutputCommitCoordinator
25/03/11 00:46:13 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
25/03/11 00:46:13 INFO Utils: Successfully started service 'SparkUI' on port 4041.
25/03/11 00:46:13 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.29.104:4041
25/03/11 00:46:13 INFO Executor: Received shutdown signal from driver, exiting...

```

```

bdalab@saraswati:~$ hdfs dfs -ls /user/studentregno/lab9/output1
Found 2 items
-rw-r--r--  3 bdalab  supergroup          0 2025-03-11 00:46 /user/studentregno/lab9/output1/_SUCCESS
-rw-r--r--  3 bdalab  supergroup  2357 2025-03-11 00:46 /user/studentregno/lab9/output1/part-00000
bdalab@saraswati:~$ hdfs dfs -cat /user/studentregno/lab9/output1/part-00000
('One', 1)
('of', 10)
('the', 19)
('three', 1)
('components', 1)
('Hadoop', 4)
('is', 9)
('Map', 3)
('Reduce.', 1)
('The', 3)
('first', 2)
('component', 2)
('that', 7)
('is,', 2)
('Distributed', 1)
('File', 1)
('System', 1)
('HDFS', 1)
('responsible', 2)
('for', 5)
('storing', 2)
('file.', 5)
('second', 1)
('Reduce', 2)
('processing', 1)
('MapReduce', 1)
('has', 1)
('mainly', 1)
('2', 2)
('tasks', 1)
('which', 1)
('are', 5)
('divided', 2)

```

## Week-9 Exercises:

1. Process log files and extract useful insights using Spark.

- Read the server log file from HDFS into an RDD.
- Extract HTTP status codes and count occurrences.
- Find the most common IP addresses accessing the server.
- Filter out error logs (HTTP 4xx and 5xx) and count them.
- Write the results back to HDFS.

Sample input file:

```
192.168.1.1 - - [08/Mar/2025:10:30:00 +0000] "GET /index.html HTTP/1.1" 200 1024
192.168.1.2 - - [08/Mar/2025:10:31:00 +0000] "POST /login HTTP/1.1" 302 512
192.168.1.3 - - [08/Mar/2025:10:32:00 +0000] "GET /home HTTP/1.1" 200 2048
192.168.1.1 - - [08/Mar/2025:10:33:00 +0000] "GET /about.html HTTP/1.1" 404 256
192.168.1.4 - - [08/Mar/2025:10:34:00 +0000] "GET /contact.html HTTP/1.1" 500 128
192.168.1.5 - - [08/Mar/2025:10:35:00 +0000] "GET /services.html HTTP/1.1" 200 3072
192.168.1.2 - - [08/Mar/2025:10:36:00 +0000] "GET /index.html HTTP/1.1" 200 1024
192.168.1.3 - - [08/Mar/2025:10:37:00 +0000] "GET /login HTTP/1.1" 403 512
192.168.1.4 - - [08/Mar/2025:10:38:00 +0000] "POST /api/v1/data HTTP/1.1" 201 4096
192.168.1.5 - - [08/Mar/2025:10:39:00 +0000] "GET /dashboard HTTP/1.1" 200 5120
192.168.1.1 - - [08/Mar/2025:10:40:00 +0000] "POST /upload HTTP/1.1" 503 256
```

2. Work with structured data in JSON format using Spark.

- Read a JSON file from HDFS into a Spark DataFrame.
- Select specific fields and display them.
- Filter records based on a condition (e.g., users older than 30).
- Perform a group-by operation on a categorical field.
- Save the processed data as a new JSON file in HDFS.

Sample input JSON file:

```
[{"name": "Alice", "age": 25, "city": "New York"}, {"name": "Bob", "age": 35, "city": "Los Angeles"}, {"name": "Charlie", "age": 40, "city": "Chicago"}, {"name": "David", "age": 22, "city": "New York"}, {"name": "Eve", "age": 30, "city": "Los Angeles"}]
```

3. Process a sales dataset using Apache Spark to perform various transformations and aggregations.

- Load a CSV file from HDFS into a Spark DataFrame.
- Compute total revenue per product (Revenue = Price \* Quantity).
- Find the highest-selling product (by revenue).

- Filter transactions where total sales are above \$500.
- Group by category and compute total revenue per category.
- Save the transformed DataFrame back to HDFS in CSV format.

Sample input:

```
TransactionID,Date,Category,Price,Quantity
1001,2024-03-01,Laptop,Electronics,800,5
1002,2024-03-02,Smartphone,Electronics,500,10
1003,2024-03-03,Headphones,Accessories,50,15
1004,2024-03-04,Monitor,Electronics,300,7
1005,2024-03-05,Mouse,Accessories,25,20
```