EXP -6

For hadoop.3x version http://localhost:9870 http://localhost:8088/cluster

Place the mapper file ,reducer file and hadoop dtearming jar file in Documents,Create input foler in hadoop and place the wordcount.txt file on it.

hadoop/Documents$ give below comments to run hadoop@Ubuntu:~/Documents$ hadoop jar hadoop-streaming-2.7.3.jar -input

/home/hadoop/input/word\_count\_data.txt -output /home/hadoop/output -mapper mapper.py

reducer reducer.py

To check the output folder part-oooo file is created or not hadoop@Ubuntu:~/Documents$ hadoop fs -ls /home/hadoop/output

hadoop@Ubuntu:~/Documents$ hadoop fs -ls /home/hadoop/output Found 2 items

-rw-r--r-- 1 hadoop supergroup 0 2024-08-03 08:59 /home/hadoop/output/\_SUCCESS

-rw-r--r-- 1 hadoop supergroup 592 2024-08-03 08:59 /home/hadoop/output/part-00000

hadoop@Ubuntu:~/Documents$ hdfs dfs -cat /user/hadoop/output/part-00000 cat: `/user/hadoop/output/part-00000': No such file or directory

## Verify the output

hadoop@Ubuntu:~/Documents$ hdfs dfs -cat /home/hadoop/output/part-00000 2,000 1

ChatGPT 1

Did 1

Roman 2

Romans 1

Some 1

Sure! 1

This 1

a 3

actually 1

ancient 1 and

3

ash 1 ash,

1

because

1

buildings

1

called 1

concrete 2 concrete, 1

concrete. 1

construction 1 durable

1 for 1 form 2 from 1

gets 1 harbors 1

has 1 have 1

impressive 1

incredibly 1

is 1 know

1 lime, 1 longevity

1 made

1 many 1 mineral

1

mixture

1

modern

1 of

3

outlasted

1. over
2. partly

1 reacts 1 reinforces

1

seawater

1

seawater,

1

showcasing

1

stronger

1

structures?

1

survived

1

techniques

1

that 2

the 5

their 1

time. 1 to

1

tobermorite, 1

used 1

volcanic 2

was 1

which 1

with 1

years, 1 you

1

path of hadoop input file

/home/hadoop/input/word\_count\_data.txt

path of hadoop output file /home/hadoop/output

Commands on hadoop to check the input and output file

# List Contents of a Hadoop Directory

To list the contents of a directory in HDFS, use the -ls option with the hadoop fs command.

List the Input Folder

bash

hadoop fs -ls /user/hduser/input

List the Output Folder

bash

hadoop fs -ls /user/hduser/output

1. View Detailed Information

The -ls command provides a detailed listing of files and directories, including permissions, owner, group, size, and modification date.

1. View File Contents

To view the contents of a file, use the -cat option. For example:

View a File in the Input Folder

bash

hadoop fs -cat /user/hduser/input/filename.txt

Replace filename.txt with the actual name of the file you want to view. View a File in the Output Folder

If your output folder contains multiple files (e.g., part-r-00000), you can view one of the files:

bash

hadoop fs -cat /user/hduser/output/part-r-00000

1. Check for Folder Existence

To check if a folder exists in HDFS, you can use the -test command with the -d option:

Check if Input Folder Exists

bash

hadoop fs -test -d /user/hduser/input && echo "Input folder exists" || echo "Input folder does not exist"

Check if Output Folder Exists

bash

hadoop fs -test -d /user/hduser/output && echo "Output folder exists" || echo "Output folder does not exist"

1. Additional Commands

Make a Directory:

bash

hadoop fs -mkdir /user/hduser/new\_directory

Delete a File or Directory:

bash

hadoop fs -rm /user/hduser/input/filename.txt hadoop fs -rm -r /user/hduser/output

Get File Status:

bash

hadoop fs -stat /user/hduser/input/filename.txt

Example Usage

Assuming you have the following setup:

Input Folder: /user/hduser/input Output Folder: /user/hduser/output

You would list the contents like this:

bash

hadoop fs -ls /user/hduser/input

And to check the output:

bash

hadoop fs -ls /user/hduser/output bash

hadoop fs -cat /user/hduser/output/part-r-00000

PIG UDF PROGRAM

To check the pig program

hadoop@Ubuntu:~/Documents$ nano sample.txt Paste the below content to sample.txt

1,John 2,Jane 3,Joe 4,Emma

hadoop@Ubuntu:~/Documents$ hadoop fs -put sample.txt /home/hadoop/piginput/

hadoop@Ubuntu:~/Documents$ nano demo\_pig.pig

paste the below the content to demo\_pig.pig

-- Load the data from HDFS

data = LOAD '/home/hadoop/piginput/sample.txt' USING PigStorage(',') AS (id:int>

-- Dump the data to check if it was loaded correctly DUMP data;

hadoop@Ubuntu:~/Documents$ pig demo\_pig.pig

2024-08-07 12:13:08,791 [main] INFO

org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1 (1,John)

(2,Jane)

(3,Joe)

(4,Emma)

By using these commands, you can manage and inspect files and directories in your Hadoop setup.

up

To Run pig basic program and uf program

uppercase\_udf.py

- def uppercase(text): return text.upper()

if name ==

" main ": import sys for line in sys.stdin:

line = line.strip() result

= uppercase(line) print(result)

Create the udfs folder on hadoop

hadoop@Ubuntu:~/Documents$ hadoop fs -mkdir /home/hadoop/udfs

put the upppercase\_udf.py in to the abv folder hadoop@Ubuntu:~/Documents$ hdfs dfs -put uppercase\_udf.py /home/hadoop/udfs/ hadoop@Ubuntu:~/Documents$ nano udf\_example.pig **udf\_example.pig**

-- Register the Python UDF script

REGISTER 'hdfs:///home/hadoop/udfs/uppercase\_udf.py' USING jython AS udf;

-- Load some data data = LOAD 'hdfs:///home/hadoop/sample.txt' AS (text:chararray);

-- Use the Python UDF

uppercased\_data = FOREACH data GENERATE udf.uppercase(text) AS uppercase\_text;

-- Store the result

STORE uppercased\_data INTO 'hdfs:///home/hadoop/pig\_output\_data';

place sample.txt fle on hadoop hadoop@Ubuntu:~/Documents$ hadoop fs -put sample.txt /home/hadoop/

To Run the pig file

hadoop@Ubuntu:~/Documents$ pig -f udf\_example.pig

finally u get Success!

Job Stats (time in seconds):

JobId Maps Reduces MaxMapTime MinMapTime AvgMapTime MedianMapTime MaxReduceTime MinReduceTime AvgReduceTime MedianReducetime Alias Feature Outputs

job\_local1786848041\_0001 1 0 n/a n/a n/a n/a 00 0 0

data,uppercased\_data MAP\_ONLY hdfs:///home/hadoop/pig\_output\_data,

Input(s):

Successfully read 4 records (42778068 bytes) from: "hdfs:///home/hadoop/sample.txt"

Output(s):

Successfully stored 4 records (42777870 bytes) in: "hdfs:///home/hadoop/pig\_output\_data"

Counters:

Total records written : 4

Total bytes written : 42777870

Spillable Memory Manager spill count : 0 Total bags proactively spilled: 0

Total records proactively spilled: 0

Job DAG:

job\_local1786848041\_0001

2024-08-07 13:33:04,631 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!

2024-08-07 13:33:04,639 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!

2024-08-07 13:33:04,644 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!

2024-08-07 13:33:04,667 [main] INFO

org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!

To check the output file is created

hadoop@Ubuntu:~/Documents$ hdfs dfs -ls /home/hadoop/pig\_output\_data Found 2 items

If you need to examine the files in the output folder, use:

To view the output

hadoop@Ubuntu:~/Documents$ hdfs dfs -cat /home/hadoop/pig\_output\_data/part-m-00000

## bash: put the employees.json local directory to *home/*hadoop directory

pig shell: Load the json file by giving following command

grunt>-- Load the data employees = LOAD '/home/hadoop/emp.json' USING

JsonLoader('name:chararray,age:int,department:chararray,salary:float'); grunt>projected

= FOREACH employees GENERATE name, salary; DUMP projected;

1. Aggregation

Aggregate the total salary:

pig

-- Load the data

employees = LOAD '/home/hadoop/employees.json' USING JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Aggregate: Calculate the total salary

total\_salary = FOREACH (GROUP employees ALL) GENERATE SUM(employees.salary) AS total\_salary;

DUMP total\_salary;

Output:

scss

(315000.0)

1. Skip

Skip the first 2 records:

pig

-- Load the data

employees = LOAD '/home/hadoop/employees.json' USING JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Skip the first 2 records skipped\_employees = LIMIT employees 1000000; -- Use LIMIT to handle skipping

DUMP skipped\_employees;

Output:

name age department salary

|  |  |
| --- | --- |
| Alice Johnson 35 | Finance 70000 |
| Bob Brown 28 | Marketing 55000 |
| Charlie Black 45 | IT 80000 |

Note: The LIMIT command should be used with an appropriate number, as Pig does not directly support skipping a specific number of records.

1. Limit

Limit the results to the top 3 records:

pig

-- Load the data

employees = LOAD '/home/hadoop/employees.json' USING JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Limit: Get the top 3 highest earners top\_3\_employees = LIMIT employees 3;

DUMP top\_3\_employees;

Output:

name age department salary

|  |  |
| --- | --- |
| Charlie Black 45 | IT 80000 |
| Alice Johnson 35 | Finance 70000 |
| Jane Smith 25 | IT 60000 |

1. Count

Count the number of employees:

pig

-- Load the data

employees = LOAD '/home/hadoop/employees.json' USING JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Count the number of employees

employee\_count = FOREACH (GROUP employees ALL) GENERATE COUNT(employees) AS total\_count;

DUMP employee\_count;

Output:

scss

(5)

1. Remove

Remove employees from a specific department, e.g., "IT":

pig

-- Load the data

employees = LOAD '/home/hadoop/employees.json' USING JsonLoader('name:chararray,age:int,department:chararray,salary:float');

-- Remove employees from the 'IT' department filtered\_employees = FILTER employees BY department != 'IT';

DUMP filtered\_employees;

|  |  |  |
| --- | --- | --- |
| Output:  name age | department | salary |
| John Doe | 30 HR | 50000 |

Alice Johnson 35 Finance 70000

Bob Brown 28 Marketing 55000

============================================================

## import Json file and do projetion, aggregation, limit,count ,skip and remove using python and hdfs

Steps to be followed:

**Install** pandas and hdfs using pip.

* **Optionally** install pyarrow or hdfs3 if needed based on your specific requirements.
* **Verify** the installation to ensure everything is set up correctly.

# Required Packages

pandas:

Purpose: Provides data structures and functions to efficiently manipulate and analyze data. Installation: Use pip to install pandas.

bash

pip install pandas

hdfs:

Purpose: Provides a Python interface to interact with HDFS. Installation: Use pip to install hdfs.

bash

pip install hdfs

Additional Considerations

While the script should work with just the above packages, here are some additional considerations:

pyarrow (Optional but useful):

Purpose: If you're working with Apache Arrow or need additional features for handling large datasets or different file formats, pyarrow can be useful.

Installation: Use pip to install pyarrow. bash

pip install pyarrow

hdfs3 (Alternative to hdfs):

Purpose: Another Python library for interacting with HDFS. It's an alternative to the hdfs package and might be preferred in some scenarios.

Installation: Use pip to install hdfs3.

bash

pip install hdfs3

Verifying Package Installation

After installing the required packages, you can verify that they are correctly installed and accessible in your Python environment:

python

import pandas as pd from hdfs import InsecureClient

# Check pandas version

print("Pandas version:", pd. version )

# Test HDFS client connection client = InsecureClient('http://localhost:9870', user='hadoop') print("HDFS status:", client.status('/'))

If you run this script and see the version of pandas and a status message from HDFS without any errors, the packages are installed correctly.

Create process\_data.py file from hdfs import InsecureClient import pandas as pd import json

# Connect to HDFS hdfs\_client = InsecureClient('http://localhost:9870', user='hdfs')

# Read JSON data from HDFS try: with hdfs\_client.read('/home/hadoop/emp.json', encoding='utf-8') as reader:

json\_data = reader.read() # Read the raw data as a string if not json\_data.strip(): # Check if data is empty raise ValueError("The JSON file is empty.")

print(f"Raw JSON Data: {json\_data[:1000]}") # Print first 1000 characters for debugging data = json.loads(json\_data) # Load the JSON data except json.JSONDecodeError as e: print(f"JSON Decode Error: {e}") exit(1) except Exception as e:

print(f"Error reading or parsing JSON data: {e}") exit(1)

# Convert JSON data to DataFrame try:

df = pd.DataFrame(data) except ValueError as e:

print(f"Error converting JSON data to DataFrame: {e}") exit(1)

# Projection: Select only 'name' and 'salary' columns projected\_df = df[['name', 'salary']]

# Aggregation: Calculate total salary total\_salary = df['salary'].sum()

# Count: Number of employees earning more than 50000 high\_earners\_count = df[df['salary'] > 50000].shape[0]

# Limit: Get the top 5 highest earners top\_5\_earners = df.nlargest(5, 'salary')

# Skip: Skip the first 2 employees skipped\_df = df.iloc[2:]

# Remove: Remove employees from a specific department filtered\_df = df[df['department'] != 'IT']

# Save the filtered result back to HDFS filtered\_json = filtered\_df.to\_json(orient='records') try:

with hdfs\_client.write('/home/hadoop/filtered\_employees.json', encoding='utf-8', overwrite=True) as writer:

writer.write(filtered\_json)

print("Filtered JSON file saved successfully.") except Exception as e:

print(f"Error saving filtered JSON data: {e}") exit(1)

# Print results

print(f"Projection: Select only name and salary columns") print(f"{projected\_df}")

print(f"Aggregation: Calculate total salary")

print(f"Total Salary: {total\_salary}") print(f"\n")

print(f"# Count: Number of employees earning more than 50000")

Filtered DataFrame

