
🔧 Step-by-Step Guide

---

1. Understand the Input Log File Format

Assume a **simple log format** like:

```
user1 login 2025-05-07 10:00:00
user1 logout 2025-05-07 12:30:00
user2 login 2025-05-07 11:00:00
user2 logout 2025-05-07 14:00:00
user1 login 2025-05-07 15:00:00
user1 logout 2025-05-07 17:00:00
```

We want to **track total login duration per user**.

---

2. Create the Java MapReduce Code

You'll need three main classes:

- Mapper
- Reducer
- Driver

📁 **UserLoginMapper.java**

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;

public class UserLoginMapper extends Mapper<LongWritable, Text, Text,
LongWritable> {
    private HashMap<String, Long> loginTimes = new HashMap<>();
    private SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        String[] parts = value.toString().split(" ");
        String user = parts[0];
        String action = parts[1];
        String timestamp = parts[2] + " " + parts[3];

        try {
```

```
            Date date = format.parse(timestamp);
            if (action.equalsIgnoreCase("login")) {
                loginTimes.put(user, date.getTime());
            } else if (action.equalsIgnoreCase("logout") &&
loginTimes.containsKey(user)) {
                long duration = date.getTime() - loginTimes.get(user);
                context.write(new Text(user), new LongWritable(duration));
                loginTimes.remove(user);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

📁 **UserLoginReducer.java**

```
import java.io.IOException;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Reducer;

public class UserLoginReducer extends Reducer<Text, LongWritable, Text,
LongWritable> {
    @Override
    protected void reduce(Text key, Iterable<LongWritable> values, Context
context)
            throws IOException, InterruptedException {
        long totalDuration = 0;
        for (LongWritable val : values) {
            totalDuration += val.get();
        }
        context.write(key, new LongWritable(totalDuration));
    }
}
```

📁 **UserLoginDriver.java**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class UserLoginDriver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "User Login Duration");

        job.setJarByClass(UserLoginDriver.class);
        job.setMapperClass(UserLoginMapper.class);
        job.setReducerClass(UserLoginReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

---

3. Compile and Create JAR

From your Cloudera terminal:

```
mkdir logapp
cd logapp

# Copy your .java files here
javac -classpath `hadoop classpath` -d . UserLoginMapper.java
UserLoginReducer.java UserLoginDriver.java

jar -cvf loganalyzer.jar *.class
```

---

4. Prepare Input File

```
hdfs dfs -mkdir /user/cloudera/logs
hdfs dfs -put system_logs.txt /user/cloudera/logs/
```

---

5. Run MapReduce Job

```
hadoop jar loganalyzer.jar UserLoginDriver /user/cloudera/logs
/user/cloudera/logoutput
```

---

6. View Output

```
hdfs dfs -cat /user/cloudera/logoutput/part-r-00000
```

---

7. Bonus: Find the Max Login User

If you want to post-process to find **user with maximum login duration**, use:

```
hdfs dfs -cat /user/cloudera/logoutput/part-r-00000 | sort -k2 -nr | head -
1
```

🧪 Sample Output

```
user1 27000000
user2 10800000
```

Which means user1 has the highest login duration.

Would you like:

- A downloadable sample log file?
- Help with automating all this via shell script?
- To convert this into a Hive/Pig script too?

Let me know how you'd like to extend this next 😊

🔧 Setup: Open Hive Shell

```
hive
```

🧱 a. Creating, Dropping, and Altering Database Tables

✅ Step 1: Create a database

```
CREATE DATABASE flight_db;
```

🔍 Check:

```
SHOW DATABASES;
```

✅ Step 2: Use the database

```
USE flight_db;
```

✅ Step 3: Create flight_info table

```
CREATE TABLE flight_info (
    flight_id INT,
    airline STRING,
    source STRING,
    destination STRING,
    departure_time STRING,
    arrival_time STRING,
    dep_delay INT,
    flight_date DATE
);
```

🔍 Check:

```
SHOW TABLES;
DESCRIBE flight_info;
```

✅ Step 4: Alter the table

```
ALTER TABLE flight_info ADD COLUMNS (aircraft_type STRING);
```

🔍 Check again:

```
DESCRIBE flight_info;
```

✅ Step 5: Drop the table

```
DROP TABLE IF EXISTS flight_info;
```

🔍 Check:

```
SHOW TABLES;
```

🌐 b. Creating an External Hive Table

✅ Step 6: Create external table

```
CREATE EXTERNAL TABLE ext_flight_info (
    flight_id INT,
    airline STRING,
    source STRING,
    destination STRING,
    departure_time STRING,
    arrival_time STRING,
    dep_delay INT,
    flight_date STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/external/flight_info';
```

🔍 Check:

```
SHOW TABLES;
DESCRIBE FORMATTED ext_flight_info;
```

📥 c. Load Data, Insert New Values, Join Tables

✅ Step 7: Create managed table again

```
CREATE TABLE flight_info (
    flight_id INT,
    airline STRING,
    source STRING,
    destination STRING,
    departure_time STRING,
    arrival_time STRING,
```

```
    dep_delay INT,
    flight_date DATE
);
```

✅ Step 8: Load data into `flight_info` table

Ensure your CSV file is in HDFS:

```
hdfs dfs -put /home/cloudera/flight_data.csv
/user/hive/warehouse/flight_db.db/
```

Then load it:

```
LOAD DATA INPATH '/user/hive/warehouse/flight_db.db/flight_data.csv'
INTO TABLE flight_info;
```

🔍 Check data:

```
SELECT * FROM flight_info LIMIT 5;
```

✅ Step 9: Insert new value

```
INSERT INTO TABLE flight_info VALUES (
    1001, 'Indigo', 'DEL', 'BOM', '08:30', '10:30', 5, '2008-05-12'
);
```

🔍 Check:

```
SELECT * FROM flight_info WHERE flight_id = 1001;
```

✅ Step 10: Join with another table

Create another table for airport info:

```
CREATE TABLE airport_info (
    code STRING,
    name STRING,
    city STRING
);
```

Insert dummy data:

```
INSERT INTO airport_info VALUES
('DEL', 'Indira Gandhi International', 'Delhi'),
('BOM', 'Chhatrapati Shivaji Maharaj International', 'Mumbai');
```

Join example:

```
SELECT f.flight_id, f.airline, a.name AS source_airport
FROM flight_info f
```

```
JOIN airport_info a
ON f.source = a.code;
```

🔎 d. Create Index on Flight Table

Note: Hive indexing is deprecated in newer versions, but you can use it like this if supported:

```
CREATE INDEX idx_dep_delay
ON TABLE flight_info (dep_delay)
AS 'COMPACT'
WITH DEFERRED REBUILD;
```

Then rebuild:

```
ALTER INDEX idx_dep_delay ON flight_info REBUILD;
```

🔍 Check:

```
SHOW INDEX ON flight_info;
```

📊 e. Find Average Departure Delay Per Day in 2008

```
SELECT
  flight_date,
  AVG(dep_delay) AS avg_delay
FROM flight_info
WHERE YEAR(flight_date) = 2008
GROUP BY flight_date
ORDER BY flight_date;
```

🧪 Sample Output Format

```
flight_date   | avg_delay
--------------|-----------
2008-01-01    | 12.3
2008-01-02    | 15.0
...
```

Great, Chetan! Below are **Python code snippets** using **pandas, NumPy, and scikit-learn** to perform the required tasks on each dataset. You can adapt these to your exact dataset structure.

✅ 3. Facebook Metrics Dataset (Pandas Operations)

```
import pandas as pd

# Load dataset
df = pd.read_csv('facebook_metrics.csv')

# a. Create Data Subsets
subset = df[['Type', 'Category', 'Total Interactions']]
print(subset.head())

# b. Merge Data
extra = pd.DataFrame({'Type': ['Photo'], 'Extra': ['Sample']})
merged = pd.merge(df, extra, on='Type', how='left')
print(merged.head())

# c. Sort Data
sorted_df = df.sort_values(by='Total Interactions', ascending=False)
print(sorted_df.head())

# d. Transpose Data
transposed = df.head().T
print(transposed)

# e. Shape and Reshape
print("Original shape:", df.shape)
reshaped = df.values.reshape(-1, df.shape[1])
print("Reshaped shape:", reshaped.shape)
```

✅ 4. Heart Disease Dataset

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Load dataset
heart_df = pd.read_csv('heart_disease.csv')

# a. Data Cleaning (Missing values)
imputer = SimpleImputer(strategy='mean')
heart_df[['chol', 'thalach']] = imputer.fit_transform(heart_df[['chol',
'thalach']])

# b. Data Integration
extra_info = pd.read_csv('hospital_info.csv')
merged_df = pd.merge(heart_df, extra_info, on='hospital_id', how='left')

# c. Data Transformation (Normalization)
scaler = StandardScaler()
```

```
heart_df[['age', 'chol', 'thalach']] =
scaler.fit_transform(heart_df[['age', 'chol', 'thalach']])

# d. Error Correcting (e.g., fixing invalid entries)
heart_df['sex'] = heart_df['sex'].replace({2: 1})  # Assuming 2 is an
invalid entry

# e. Data Model Building
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X = heart_df.drop('target', axis=1)
y = heart_df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

✅ 5. Air Quality Dataset

```
import pandas as pd
import numpy as np

# Load data
air_df = pd.read_csv('air_quality.csv')

# a. Data Cleaning
air_df.dropna(inplace=True)

# b. Data Integration
weather_df = pd.read_csv('weather_data.csv')
merged_air = pd.merge(air_df, weather_df, on='date', how='left')

# c. Data Transformation
air_df['PM2.5_log'] = np.log1p(air_df['PM2.5'])

# d. Error Correcting
air_df.loc[air_df['PM2.5'] < 0, 'PM2.5'] = air_df['PM2.5'].mean()
```
