PROBLEM STATEMENT: Design a distributed application using MapReduce which processes a log file of a system. List out the users who have logged for maximum period on the system. Use simple log file from the Internet and process it using a pseudo distribution mode on Hadoop platform.

Step 1: Create a Java Project in Eclipse

1. **Create a New Java Project**:
    - Open Eclipse and go to **File** → **New** → **Java Project**.
    - Name the project (e.g., `HadoopLogFileProcessor`).
    - Click **Finish**.
2. **Create a Package**:
    - In the **Project Explorer**, right-click on `src` → **New** → **Package**.
    - Name the package (e.g., `HadoopLogFileProcessor`).
3. **Create the Classes**:
    - Right-click on the package you created → **New** → **Class**.
    - Create the following classes:
        - **LoginMapper**
        - **LoginReducer**
        - **ProcessLogFile** (Driver class)

Step 2: Add the Code

Add the following code in their respective classes:

//LoginMapper.java

```
package HadoopLogFileProcessor;


import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;
```

```
import java.io.IOException;


public class LoginMapper extends Mapper<Object, Text, Text, IntWritable> {


    // For emitting user login and logout events

    private Text user = new Text();

    private IntWritable timestamp = new IntWritable();


    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

        String line = value.toString();

        String[] parts = line.split(" ");

     // Extract timestamp, user, and action

        int time = Integer.parseInt(parts[0]);

        String username = parts[1];

        String action = parts[2];


        if (action.equals("login")) {

            // Emit the login timestamp with key as <username>_login

            user.set(username + "_login");

            timestamp.set(time);

            context.write(user, timestamp);

        } else if (action.equals("logout")) {

            // Emit the logout timestamp with key as <username>_logout

            user.set(username + "_logout");

            timestamp.set(time);

            context.write(user, timestamp);
```

```
        }
    }
}
```

//LoginReducer.java

```
package HadoopLogFileProcessor;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class LoginReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private Map<String, Integer> userSessionDurations = new HashMap<>();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        String username = key.toString().split("_")[0];
        // Iterate through all timestamps for login/logout
        int loginTimestamp = -1;
        int logoutTimestamp = -1;

        for (IntWritable val : values) {
            if (key.toString().endsWith("login")) {
```

```
            loginTimestamp = val.get();
        } else if (key.toString().endsWith("logout")) {
            logoutTimestamp = val.get();
        }
    }

    // Calculate session duration if both login and logout are present
    if (loginTimestamp != -1 && logoutTimestamp != -1) {
        int sessionDuration = logoutTimestamp - loginTimestamp;
        userSessionDurations.put(username, sessionDuration);
    }
}

// Output the user with the longest session
@Override
protected void cleanup(Context context) throws IOException, InterruptedException {
    String maxUser = "";
    int maxDuration = 0;

    for (Map.Entry<String, Integer> entry : userSessionDurations.entrySet()) {
        if (entry.getValue() > maxDuration) {
            maxDuration = entry.getValue();
            maxUser = entry.getKey();
        }
    }
    context.write(new Text("User with longest session: " + maxUser), new
IntWritable(maxDuration));
```

```
    }
}
```

//ProcessLogFile.java

```
package HadoopLogFileProcessor;


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




public class ProcessLogFile {


    public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Log File Processor");


        // Set the Jar class
        job.setJarByClass(ProcessLogFile.class);


        // Set Mapper and Reducer classes
        job.setMapperClass(LoginMapper.class);
```

```
        job.setReducerClass(LoginReducer.class);

        // Set output key and value types

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);


        // Set input and output paths

        FileInputFormat.addInputPath(job, new Path(args[1]));

        FileOutputFormat.setOutputPath(job, new Path(args[2]));


        // Wait for job completion

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

    }

}
```

Step 3: Build the Project

1. **Build the Project**:
    - Click on **Project** → **Build Project** in Eclipse.
    - Make sure the project compiles without any errors.
2. **Check Build Path**:
    - Go to **Project Explorer** → Right-click on your project → **Build Path** → **Configure Build Path**.
    - Ensure that all the Hadoop JAR files you added are present in the **Libraries** section.
        i. adoop-common (e.g., `hadoop-common.jar`)

        ii. Hadoop-mapreduce-client-core (e.g., `hadoop-mapreduce-client-core-2.x.x.jar`)

Step 4: Create the JAR File

1. **Export to JAR**:
    - Go to **File** → **Export**.
    - Choose **Java** → **Runnable JAR file**.
    - Choose **Launch configuration** as `ProcessLogFile`.
    - Select the destination path and name your JAR file (e.g., `Log.jar`).
    - Click **Finish**.

Step 5: Prepare the Input Files

1. **Create the Log File (`logfile.txt`)**:
    - The `logfile.txt` should be placed in the HDFS directory. Here's the content of the file:

      1624875600 user1 login

1624875700 user2 login

1624875800 user1 logout

1624875900 user2 logout

1624876000 user3 login

1624876100 user3 logout

Upload the Input File to HDFS:

- Use the Hadoop shell to upload the input file to HDFS:

hdfs dfs -put /path/to/logfile.txt /user/cloudera/logfile.txt

Step 6: Configure the Run Configuration

1. **Set up Run Configuration**:
    - In Eclipse, go to **Run** → **Run Configurations**.
    - Select **Java Application** and click **New**.

- In the **Main** tab, select the **Project** (your Hadoop project) and the **Main Class** (`ProcessLogFile`).

Step 8: Run the Job

1. **Run the Hadoop Job**:
   [cloudera@quickstart ~]$ hadoop jar /home/cloudera/Log.jar HadoopLogFileProcessor.ProcessLogFile /user/cloudera/logfile.txt /user/cloudera/dir51
2. hdfs dfs -ls /user/cloudera/dir51

```
                Bytes Written=30
[cloudera@quickstart ~]$ hdfs dfs -ls /user/cloudera/dir51

Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2025-04-26 23:53 /user/cloudera/dir51/_SUCCESS
-rw-r--r--   1 cloudera cloudera         30 2025-04-26 23:53 /user/cloudera/dir51/part-r-00000
[cloudera@quickstart ~]$
```