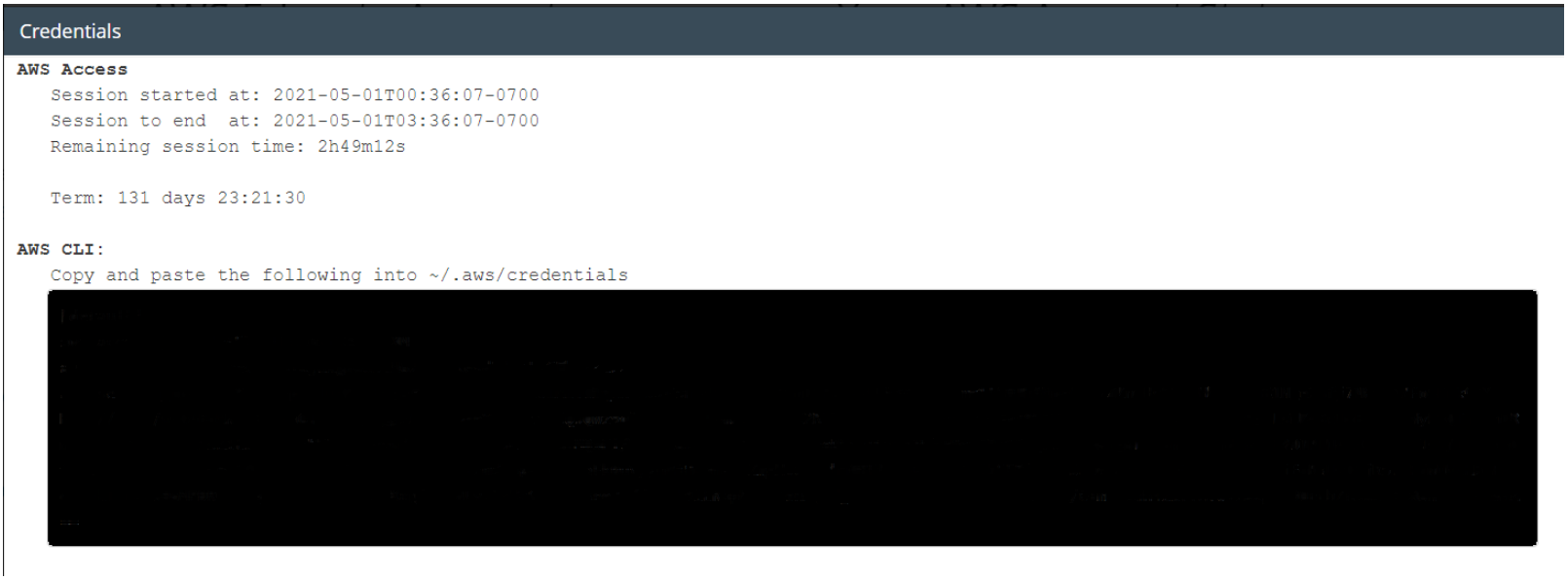
ASSIGNMENT 2 : COLLOCATION EXTRACTION

- **Running the program:**

1. You need to have an AWS EC2 account.
2. Create a file “credentials” in the path :
   C:\Users\USERNAME\.aws\credentials on Windows or
   ~/.aws/credentials on Linux, macOS, or Unix.
3. Go to your AWS account details and copy all the text inside the black box into the credentials file.



4. Set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

   To set these variables on Linux, macOS, or Unix, use **`export`** :

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

   To set these variables on Windows, use **`set`** :

```
set AWS_ACCESS_KEY_ID=your_access_key_id
```

```
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

5. The EC2 instances fetch their scripts from s3, therefore we need to make sure the jars are in the right buckets in s3 (dsps212-artifacts/jars/Ass2).

   By running the following maven builds:
   From the maven projects:

- JobStep1 -> Plugins -> assembly -> (2 clicks on) assembly: assembly
- JobStep1 -> Plugins -> s3-upload -> (2 clicks on) s3-upload:s3-upload
- Do the same for each of the other steps (JobStep2, JobStep3, JobStep4, JobStep5).

6. Run the program by executing:
   >java -jar DSP_Ass2.jar <*minPmi*> <*relMinPmi*> [optionals: <-p> <-d> <-a>]
   Or
   Run the project's Main from your IDE.
   (-p and -d are for debugging purposes, -a is for local aggregation activation).

Specs and benchmarks:

- Instance types and AMI's: m5.xlarge with default roles.
- We are using an IAM role with permissions for EC2, S3, SQS, and role transfer.
- We used 5 EC2 instances.
- We run the program twice – with and without aggregation optimization.
- The run without aggregation took 17 minutes in total and 9 minutes for the steps alone.
- The run with aggregation took 17 minutes in total and 11 minutes for the steps alone.
- These results might be due to the overhead computation caused by adding combiners in each instance and the insufficient deduction of records sent between instances or bad serialization.

- **The program flow:**

STEP 1
map
in: *{lineID -> w1w2, c(w1w2) per year, year}*
out: *{decade, w1w2 -> c(w1w2) per year}*
reduce
in: *{decade, w1w2 -> [c(w1w2) per year]}*
- compute c(w1w2) per decade.
out: *{decade, w1w2 -> c(w1w2) per decade}*

STEP 2
map
in: *{decade, w1w2 -> c(w1w2) per decade}*
out: *{tag[N/first/second/NGram], decade, id["_"/w1/w2/w1w2] -> w1w2}X4*
{tag[N/first/second/NGram], decade, id["_"/w1/w2/w1w2], "" -> c(tag) occurrences} X4*
reduce
in: *{tag[N/first/second/NGram], decade, ["_"/w1/w2/w1w2], "*" -> [c(tag) occurrences]}*
- compute c(tag) occurrences per decade.
{tag[N/first/second/NGram], decade, ["_"/w1/w2/w1w2] -> [w1w2]}
out: *[{decade, w1w2 -> tag, c(tag) occurrences per decade}]*

Step 3
reduce
in: *decade, w1w2 -> [tag[N/first/second/NGram], c(N/w1/w2/w1w2)]*
- compute npmi for w1w2.
out: *decade -> npmi, w1w2*

Step 4
map
in: *{decade -> npmi, w1w2}*
out: *{decade -> npmi, w1w2}*
{decade-0.5 -> npmi}
reduce
in: *decade-0.5 -> [npmi]*
- compute sumNPMI for decade.
decade -> [npmi, w1w2]
out: *[decade (as year) -> w1w2, sumNPMI, npmi]*

Step 5
map
in: *decade (as year) -> w1w2, sumNPMI, npmi*

- filter collocations by minPMI and relMinPMI
out: *decade (as year), npmi -> w1w2*

reduce
in: *decade (as year), npmi -> w1w2*
out: *decade (as year) -> npmi, w1w2*