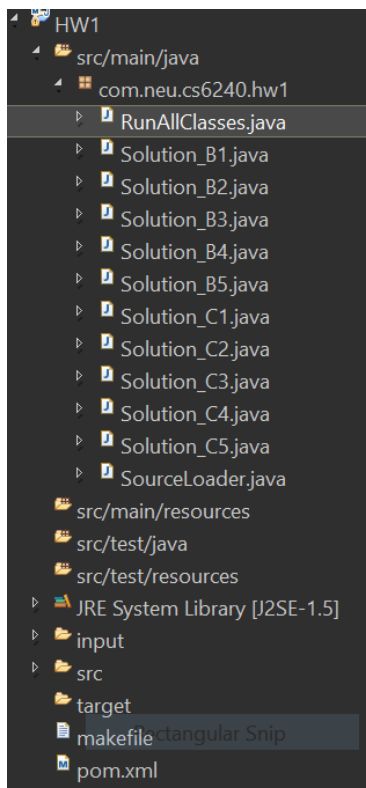


Deliverables:

1. Source Code: As explained here
2. Report: This file
3. Log files: Please find the folder log and logs in the current folder.
4. Final result AWS: Please find the folder output with the final results of the AWS run. This folder also contains the output produced by weather data source code.

Weather Data Source Code

Kindly find the source code in HW1 folder. It is a maven project with the following structure:



To compile the entire project just type in “make” once you are inside the HW1 folder.

To run the project kindly type inside the HW1 folder

- **make local filename=[fileName of the file which will be used as input]**

This will run the RunAllClasses file which will run all the rest of the classes in the project.

To run the emr cluster kindly type inside the HW1 folder

- **make emr jarFile=[Jar which needs to be run] inputFile=[Input file on which the jar needs to do the processing]**

This will run the aws emr for the given jarfile and the given inputfile.

This is all that is needed to be run the project and the emr.

PS: for emr I used --profile setting to set my aws cli profile.

Also if you want you can run individual files but kindly provide the input file name as argument.

Weather Data Results

Minimum, Maximum & Average Running Time

All running times are in milliseconds

B.1. Sequential Run

Minimum Running Time	Maximum Running Time	Average Running Time
2601.0	5953.0	3024.0

B.2. No Lock

Minimum Running Time	Maximum Running Time	Average Running Time
1314.0	2111.0	1475.0

B.3. Coarse Lock

Minimum Running Time	Maximum Running Time	Average Running Time
2936.0	6075.0	3296.0

B.4. Fine Lock

Minimum Running Time	Maximum Running Time	Average Running Time
1445.0	2569.0	1617.0

B.5. No sharing

Minimum Running Time	Maximum Running Time	Average Running Time
1327.0	4626.0	1709.0

C.1. Sequential Run

Minimum Running Time	Maximum Running Time	Average Running Time
2687.0	5767.0	3035.0

C.2. No Lock

Minimum Running Time	Maximum Running Time	Average Running Time
1307.0	4561.0	1684.0

C.3. Coarse Lock

Minimum Running Time	Maximum Running Time	Average Running Time
2990.0	6274.0	3409.0

C.4. Fine Lock

Minimum Running Time	Maximum Running Time	Average Running Time
1375.0	4714.0	1761.0

C.5. No sharing

Minimum Running Time	Maximum Running Time	Average Running Time
1361.0	4626.0	1764.0

Worker Threads used and Speedup

B.1. Sequential Run

Worker Threads	Speedup
1	N/A

B.2. No Lock

Worker Threads	Speedup
4	2.05

B.3. Coarse Lock

Worker Threads	Speedup
4	0.917

B.4. Fine Lock

Worker Threads	Speedup
4	1.87

B.5. No share

Worker Threads	Speedup
4	1.769

C.1. Sequential Run

Worker Threads	Speedup
1	N/A

C.2. No Lock

Worker Threads	Speedup
4	1.80

C.3. Coarse Lock

Worker Threads	Speedup
4	0.89

C.4. Fine Lock

Worker Threads	Speedup
4	1.723

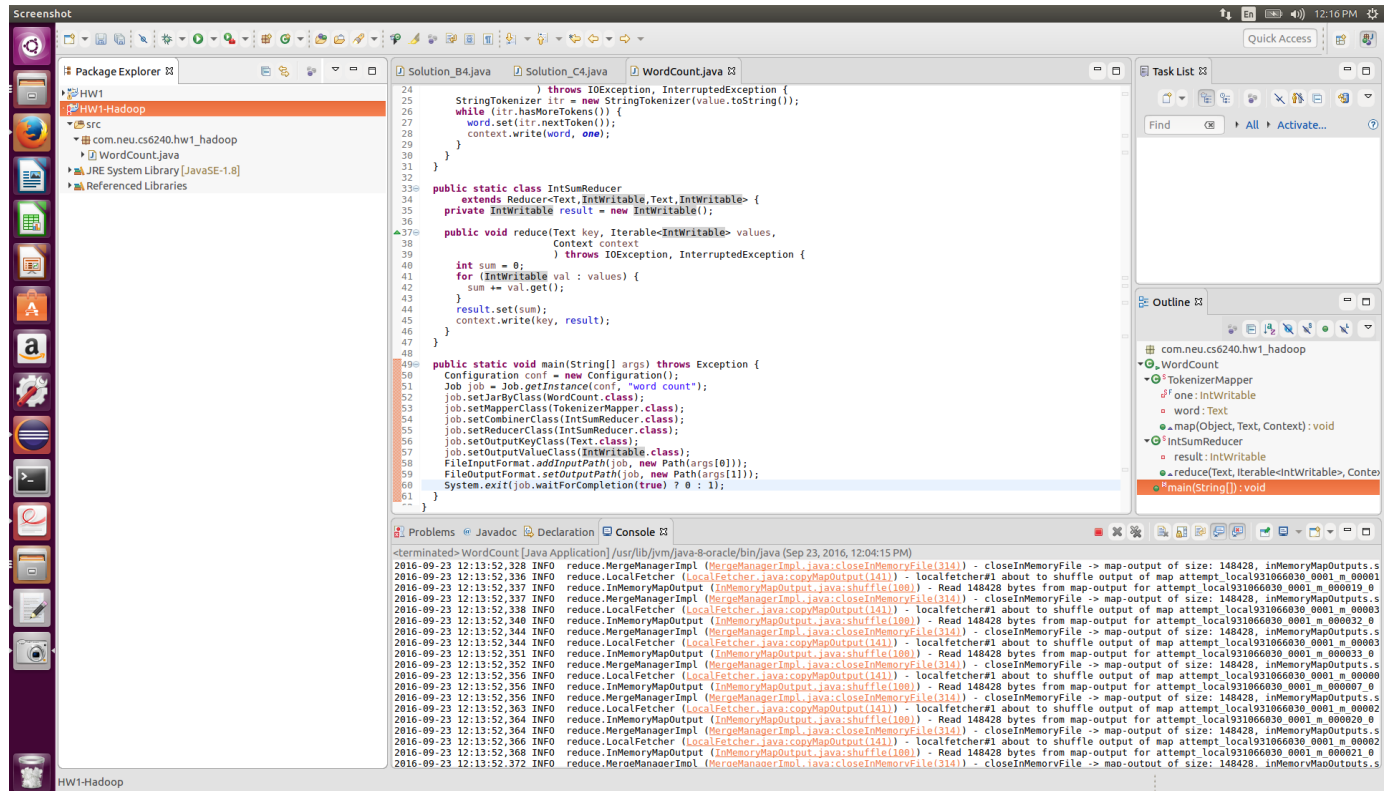
C.5. No share

Worker Threads	Speedup
4	1.720

1. The No-Lock version should end the fastest as all the threads can access the data structure at the same time. This ensures that the No-Lock will be fastest but the result obtained will not be correct. The experiments conform to this.
2. The Coarse-Lock version should end the slowest because in this case the data structure is getting locked whenever one of the threads get access to it. Once this thread gets done with the data structure it notifies other threads that they can now work on it, causing delays. The experiments conform to this.
3. The average temperatures returned by No-Lock returns incorrect values. The reason behind this is because all the threads can simultaneously update the data structure hence causing data errors.
4. Coarse Lock is slower since once a thread gets access to the data structure it blocks access to the data structure for other threads. Once it gets done with its work, it notifies other threads that they can now use the data structure. This waking up of threads causes delays. Whereas in Sequential run, only one thread works on the data structure and hence no delays are caused due to blocking.
5. The extra computation in C slows down the Fine-locking a bit as compared to Coarse-Locking. This is because for Fine-locking the Fibonacci computation happens inside the lock thereby limiting the access to the data structure whereas in Coarse-Locking in B or C the lock is on the entire data structure thereby not affecting its performance.

Word Count Local Execution

1. Project Structure:

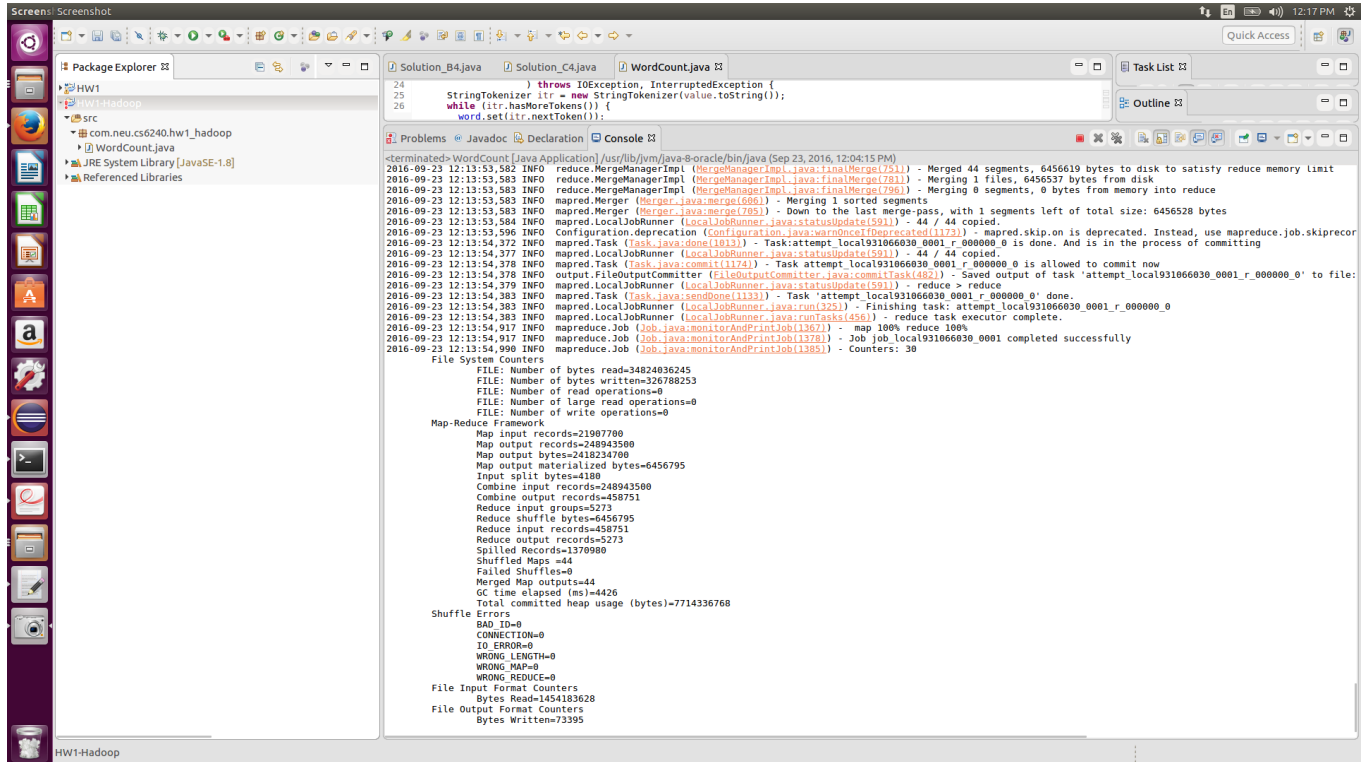


Ankit Sharma

Homework 1

CS6240 Section-01

2. Console Output:



```
<terminated> WordCount [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Sep 23, 2016, 12:04:15 PM)
2016-09-23 12:13:53,582 INFO reduce.MergeManagerImpl (MergeManagerImpl.java:finalMerge(751)) - Merged 44 segments, 6456619 bytes to disk to satisfy reduce memory limit
2016-09-23 12:13:53,583 INFO reduce.MergeManagerImpl (MergeManagerImpl.java:finalMerge(781)) - Merging 1 files, 6456537 bytes from disk
2016-09-23 12:13:53,583 INFO reduce.MergeManagerImpl (MergeManagerImpl.java:finalMerge(796)) - Merging 0 segments, 0 bytes from memory into reduce
2016-09-23 12:13:53,583 INFO mapred.Merger (Merger.java:merge(686)) - Merging 1 sorted segments
2016-09-23 12:13:53,583 INFO mapred.Merger (Merger.java:merge(705)) - Down to the last merge-pass, with 1 segments left of total size: 6456528 bytes
2016-09-23 12:13:53,584 INFO mapred.LocalJobRunner (LocalJobRunner.java:statusUpdate(591)) - 44 / 44 copied
2016-09-23 12:13:53,596 INFO Configuration.deprecation (Configuration.java:warnOnceIfDeprecated(1173)) - mapred.skip.on is deprecated. Instead, use mapreduce.job.skiprecords
2016-09-23 12:13:54,372 INFO mapred.Task (Task.java:done(1013)) - Task:attempt_local931066030_0001_r_0000000_0 is done. And is in the process of committing
2016-09-23 12:13:54,377 INFO mapred.LocalJobRunner (LocalJobRunner.java:statusUpdate(591)) - 44 / 44 copied
2016-09-23 12:13:54,378 INFO mapred.Task (Task.java:commit(1174)) - Task attempt_local931066030_0001_r_0000000_0 is allowed to commit now
2016-09-23 12:13:54,378 INFO output.FileOutputCommitter (FileOutputCommitter.java:commitTask(482)) - Saved output of task 'attempt_local931066030_0001_r_0000000_0' to file:
2016-09-23 12:13:54,379 INFO mapred.LocalJobRunner (LocalJobRunner.java:statusUpdate(591)) - reduce > reduce
2016-09-23 12:13:54,383 INFO mapred.Task (Task.java:sendDone(1133)) - Task 'attempt_local931066030_0001_r_0000000_0' done.
2016-09-23 12:13:54,383 INFO mapred.LocalJobRunner (LocalJobRunner.java:run(325)) - Finishing task: attempt_local931066030_0001_r_0000000_0
2016-09-23 12:13:54,383 INFO mapred.LocalJobRunner (LocalJobRunner.java:runTask(356)) - reduce task executor complete.
2016-09-23 12:13:54,917 INFO mapreduce.Job (Job.java:monitorAndPrintJob(1367)) - map 100% reduce 100%
2016-09-23 12:13:54,917 INFO mapreduce.Job (Job.java:monitorAndPrintJob(1378)) - Job job_local931066030_0001 completed successfully
2016-09-23 12:13:54,980 INFO mapreduce.Job (Job.java:monitorAndPrintJob(1385)) - Counters: 38

File System Counters
  FILE: Number of bytes read=34824836245
  FILE: Number of bytes written=326788253
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0

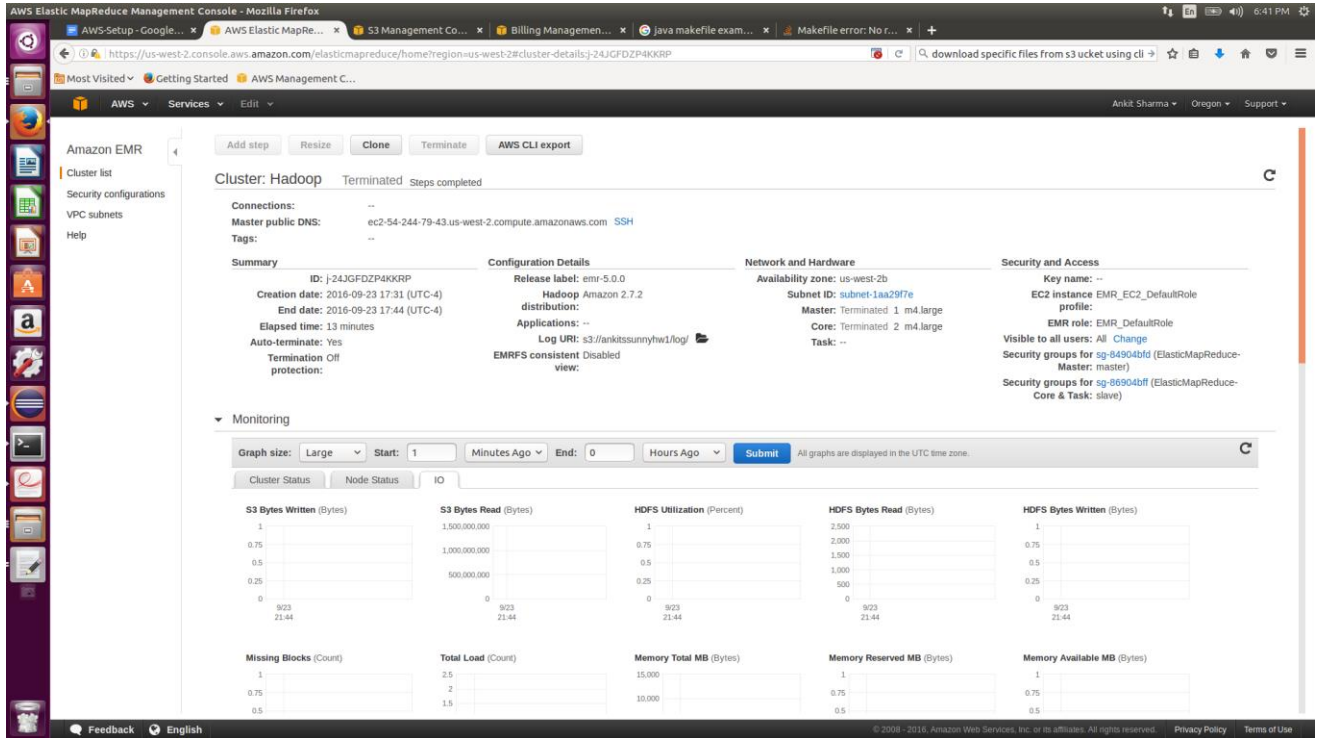
Map-Reduce Framework
  Map input records=21987700
  Map output records=248943500
  Map output bytes=2418234700
  Map output materialized bytes=6456795
  Input split bytes=4180
  Combine input records=248943500
  Combine output records=458751
  Reduce input groups=5273
  Reduce shuffle bytes=6456795
  Reduce input records=458751
  Reduce output records=5273
  Spilled Records=1370980
  Shuffled Maps =44
  Failed Shuffles=0
  Merged Map outputs=44
  GC time elapsed (ms)=4426
  Total committed heap usage (bytes)=7714336768

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=1454183628
File Output Format Counters
  Bytes Written=73395
```

Word Count AWS Execution

1. Screenshot of successful run



2. Find the log files and the output in the Assignment Folder.