Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 1
Name: Neha Pradhan

ASSIGNMENT – 1 REPORT

Weather Data Results:

- The min time, max time and the average time observes for versions of the sequential and multi-threaded program is tabulated below:

Program	Normal Run			Expensive Run		
	Min Time	Max Time	Average Time	Min Time	Max Time	Average Time
Sequential	2711	3672	2849	14406	16475	15106
No Lock	1579	1832	1677	8192	11742	9172
Coarse Lock	1678	1795	1730	14230	14854	14412
Fine Lock	1636	1831	1696	8467	9970	9177
No Sharing	1648	2094	1724	8213	10234	8587

Table 1.1

- The number of worker threads used: 2
  Speedup values for different multi-threaded versions is provided below:
  1. No Lock: 1.6988
  2. Coarse Lock: 1.6468
  3. Fine Lock: 1.6798
  4. No Sharing: 1.6525

- **Questions**

1. Which program version (SEQ, NO-LOCK, COARSE-LOCK, FINE-LOCK, NO-SHARING) would you normally expect to finish fastest and why? Do the experiments confirm your expectation? If not, try to explain the reasons.

   Multi-threaded programs allow parallel processing and hence are expected to be faster than sequential programs. Out of the four versions of multi-threaded programs, NO-LOCK would be expected to finish first as there are no locks on the accumulator data structure which would otherwise result in one thread waiting for another thread to release the locked resource.
   The results of ten runs of the programs found in Table 1.1 validates the observation that the NO-LOCK version of multi-threaded program is the fastest.

2. Which program version (SEQ, NO-LOCK, COARSE-LOCK, FINE-LOCK, NO-SHARING) would you normally expect to finish slowest and why? Do the experiments confirm your expectation? If not, try to explain the reasons

As discussed in the answer to the first question, a sequential program is always expected to run slower than a multi-threaded program as it does not allow for parallel processing. Whereas in different versions of multi-threaded programs there are two threads working on the input data processing them parallelly, sequential programs have only a single thread working on and processing the input data. As seen in Table 1.1, after ten runs of the programs, the sequential program has the slowest finish time.

3. Compare the temperature averages returned by each program version. Report if any of them is incorrect or if any of the programs crashed because of concurrent accesses.

   The temperature averages for each station is computed to have the same value in different versions of the program, except for NO-LOCK which might result in inconsistent or wrong results as there are no locks defined for the accumulator data structure shared between the threads. This allows for situations where the concurrent access and update of entries in the data structure by parallelly working threads end up creating wrong results when both threads try to update the same entry at the same time causing one thread's update to be overwritten by the other. One more observation made during separate run of the NO-LOCK program was that there were exceptions thrown due to concurrent accesses. However, this wasn't observed during the final ten run.

4. Compare the running times of SEQ and COARSE-LOCK. Try to explain why one is slower than the other. (Make sure to consider the results of both B and C—this might support or refute a possible hypothesis.)

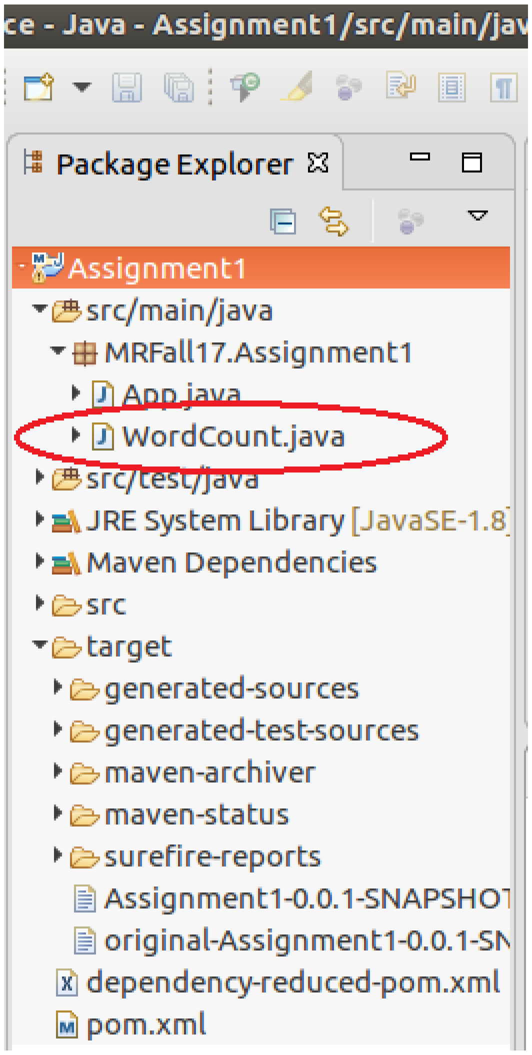
   As discussed earlier, compared to any version of multi-threaded programs sequential programs will always be slower as they employ a single thread for processing input and producing results, whereas multi-threaded programs as the name suggests has two or more threads that work parallelly on the same input and therefore producing results faster than a sequential program. The COARSE-LOCK version might not be as fast as the NO-LOCK of FINE-LOCK versions but is still faster than the SEQ version. As seen in Table 1.1, the SEQ version takes an average time of 2849ms and 15106ms to complete during the normal run and expensive run respectively where the COARSE-LOCK version takes 1730ms and 14412ms for the normal run and expensive run respectively.

5. How does the higher computation cost in part C (additional Fibonacci computation) affect the difference between COARSE-LOCK and FINE-LOCK? Try to explain the reason.

   Intuitively, COARSE-LOCK would be slower than FINE-LOCK as it locks the entire accumulator data structure while one thread requires to update the values in the data structure requiring all other threads to wait until it has completed its operation. This same behavior is observed during the normal run as seen in Table 1.1. The higher computation cost seems to cause the difference between the run times of the two versions of the programs to increase further. This could be because while running the COARSE-LOCK version each thread now has to wait for a longer period to get access to update the accumulator data structure when the other thread holds a lock to it, while in the FINE-LOCK version the threads are independent of the each other unless they have to update the same entry in the accumulator data structure. The higher computation cost in COARSE-LOCK affects all threads when a single thread holds the lock on the data structure while the threads in FINE-LOCK program work without obstructing the other unless it has to update the same entry which is not as frequent.

Word Count Local Execution:

- Project directory structure, showing that the WordCount.java file is somewhere in the src directory.

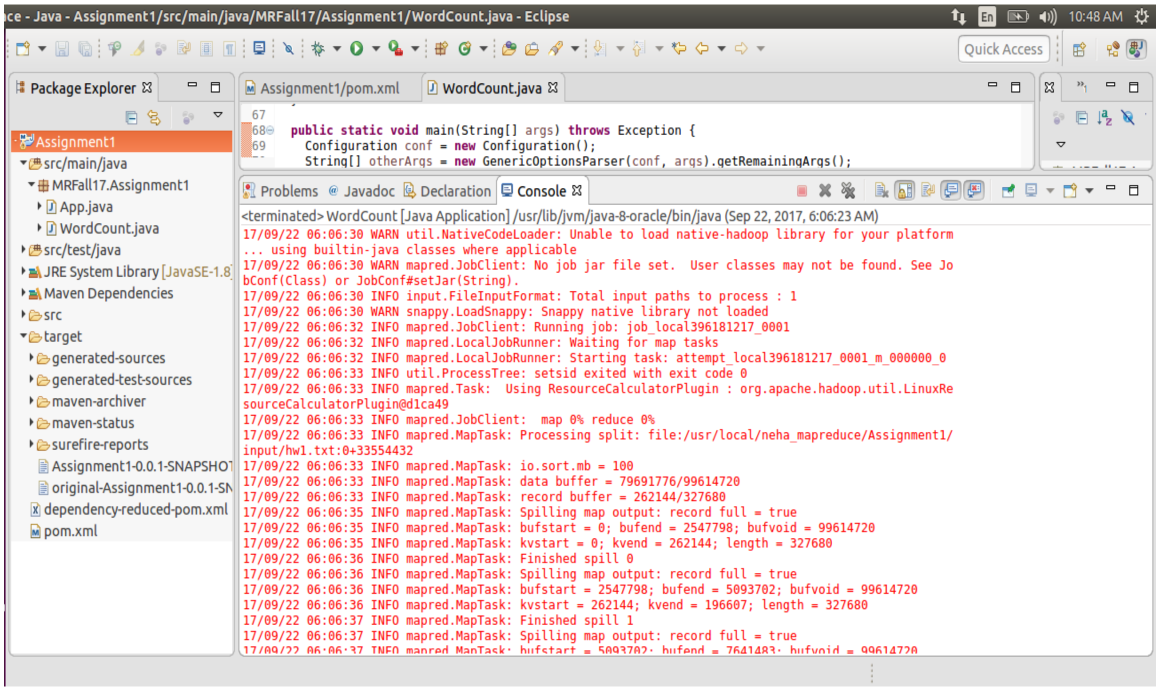


```
ce - Java - Assignment1/src/main/jav
Package Explorer
Assignment1
  src/main/java
    MRFall17.Assignment1
      App.java
      WordCount.java
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
    generated-sources
    generated-test-sources
    maven-archiver
    maven-status
    surefire-reports
    Assignment1-0.0.1-SNAPSHOT
    original-Assignment1-0.0.1-SN
  dependency-reduced-pom.xml
  pom.xml
```

Fig 1.1

- The console output for a successful run of the WordCount program inside the IDE. The console output refers to the job summary information Hadoop produces, not the output your job emits. Show at least the last 20 lines of the console output.

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 1
Name: Neha Pradhan



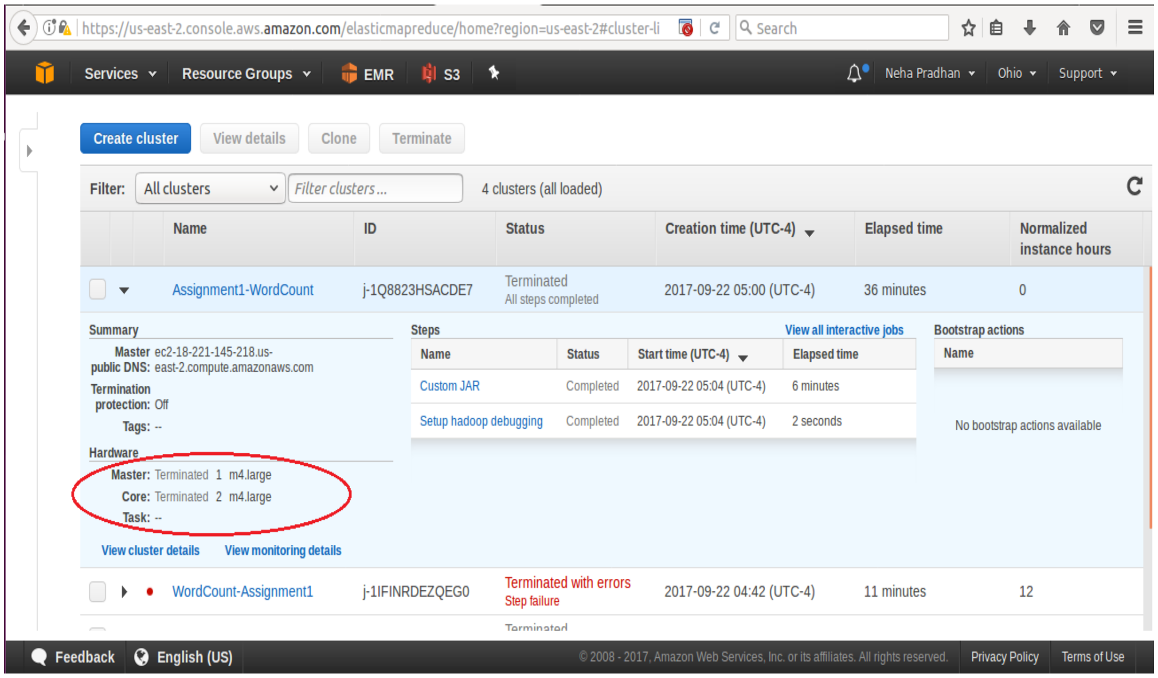
```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
```

```
<terminated> WordCount [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Sep 22, 2017, 6:06:23 AM)
17/09/22 06:06:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform
... using builtin-java classes where applicable
17/09/22 06:06:30 WARN mapred.JobClient: No job jar file set.  User classes may not be found. See Jo
bConf(Class) or JobConf#setJar(String).
17/09/22 06:06:30 INFO input.FileInputFormat: Total input paths to process : 1
17/09/22 06:06:30 WARN snappy.LoadSnappy: Snappy native library not loaded
17/09/22 06:06:32 INFO mapred.JobClient: Running job: job_local396181217_0001
17/09/22 06:06:32 INFO mapred.LocalJobRunner: Waiting for map tasks
17/09/22 06:06:32 INFO mapred.LocalJobRunner: Starting task: attempt_local396181217_0001_m_000000_0
17/09/22 06:06:33 INFO util.ProcessTree: setsid exited with exit code 0
17/09/22 06:06:33 INFO mapred.Task:  Using ResourceCalculatorPlugin : org.apache.hadoop.util.LinuxRe
sourceCalculatorPlugin@d1ca49
17/09/22 06:06:33 INFO mapred.JobClient:  map 0% reduce 0%
17/09/22 06:06:33 INFO mapred.MapTask: Processing split: file:/usr/local/neha_mapreduce/Assignment1/
input/hw1.txt:0+33554432
17/09/22 06:06:33 INFO mapred.MapTask: io.sort.mb = 100
17/09/22 06:06:33 INFO mapred.MapTask: data buffer = 79691776/99614720
17/09/22 06:06:33 INFO mapred.MapTask: record buffer = 262144/327680
17/09/22 06:06:35 INFO mapred.MapTask: Spilling map output: record full = true
17/09/22 06:06:35 INFO mapred.MapTask: bufstart = 0; bufend = 2547798; bufvoid = 99614720
17/09/22 06:06:35 INFO mapred.MapTask: kvstart = 0; kvend = 262144; length = 327680
17/09/22 06:06:36 INFO mapred.MapTask: Finished spill 0
17/09/22 06:06:36 INFO mapred.MapTask: Spilling map output: record full = true
17/09/22 06:06:36 INFO mapred.MapTask: bufstart = 2547798; bufend = 5093702; bufvoid = 99614720
17/09/22 06:06:36 INFO mapred.MapTask: kvstart = 262144; kvend = 196607; length = 327680
17/09/22 06:06:37 INFO mapred.MapTask: Finished spill 1
17/09/22 06:06:37 INFO mapred.MapTask: Spilling map output: record full = true
17/09/22 06:06:37 INFO mapred.MapTask: bufstart = 5093702; bufend = 7641483; bufvoid = 99614720
```

Fig 1.2

<u>Word Count AWS Execution:</u>

- Show a similar screenshot that provides convincing evidence of a successful run of the Word Count program on AWS. Make sure you run the program using at least three machines, i.e., one master node and two workers



Name	ID	Status	Creation time (UTC-4)	Elapsed time	Normalized instance hours
Assignment1-WordCount	j-1Q8823HSACDE7	Terminated All steps completed	2017-09-22 05:00 (UTC-4)	36 minutes	0
WordCount-Assignment1	j-1IFINRDEZQEG0	Terminated with errors Step failure	2017-09-22 04:42 (UTC-4)	11 minutes	12

Summary
Master public DNS: ec2-18-221-145-218.us-east-2.compute.amazonaws.com
Termination protection: Off
Tags: --

Hardware
Master: Terminated 1 m4.large
Core: Terminated 2 m4.large
Task: --

Steps

Name	Status	Start time (UTC-4)	Elapsed time
Custom JAR	Completed	2017-09-22 05:04 (UTC-4)	6 minutes
Setup hadoop debugging	Completed	2017-09-22 05:04 (UTC-4)	2 seconds

Bootstrap actions

Name
No bootstrap actions available

Fig 1.2