

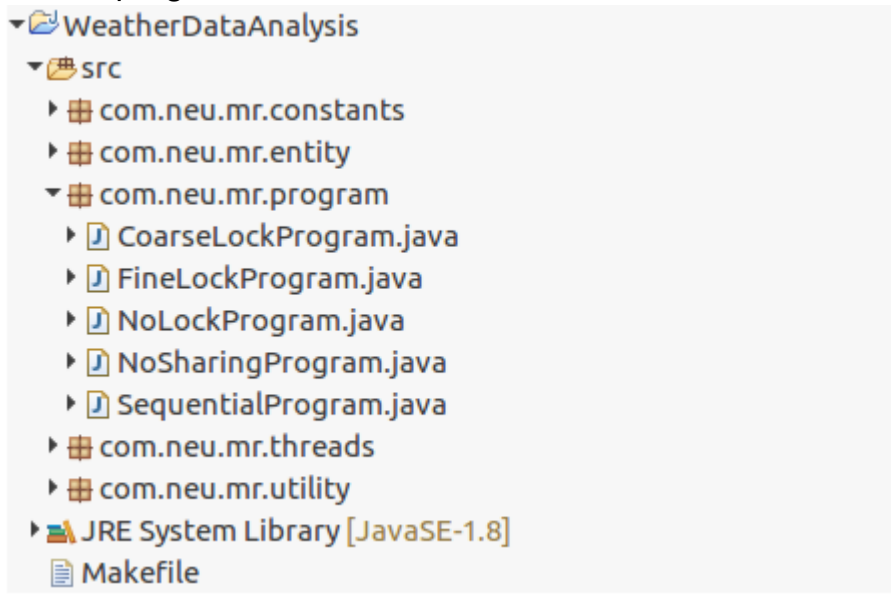
Name: Harsha Jakkappanavar

Class: CS6240 Parallel Data Processing Sec 01 Spring 2017

Homework: 1

Weather Data Source Code:

The following screen-shot depicts the directory structure for the source code of different program versions.



Weather Data Results:

The running time observed over 10 runs for each of the sequential and multithreaded programs is listed below.

	General Run			Expensive Run (Fibonacci(17))		
	Min Time	Max Time	Avg Time	Min Time	Max Time	Avg Time
Sequential	2366	2823	2479.1	11960	13298	12459
No Lock	1866	2427	2106.3	12488	12941	12675.1
Coarse Lock	1975	2591	2198.1	12355	13147	12811.8
Fine Lock	1993	2396	2168	12455	13104	12731.7
No Sharing	2141	2545	2300.9	12683	13246	12975.8

The number of threads used in multithreading is: 4

The following table depicts the speedup of the multithreaded versions based on the corresponding average running times.

	Speedup
No Lock	1.1769
Coarse Lock	1.1278
Fine Lock	1.1434
No Sharing	1.0774

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.

→ Among the 5 versions of the program, NO-LOCK version would be expected to run the fastest because all the threads are accessing the accumulation data structure concurrently because there is no lock on the accumulation data structure the threads do not have to wait. The actual results of my program execution agrees with this expectation and we can see that the NO-LOCK version of the program runs the fastest.

	Min Time	Max Time	Avg Time
No Lock	1866	2427	2106.3
Fine Lock	1993	2396	2168
Coarse Lock	1975	2591	2198.1
No Sharing	2141	2545	2300.9
Sequential	2366	2823	2479.1

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.

→ The running time of the execution of program is made up of two parts,

- Parsing the lines to find the TMAX
- Updating the accumulation datastructure.

Among the 5 versions of the program, SEQUENTIAL execution of the program runs slowest, because in the multithreaded versions of the program the accumulation data structure is always busy with one of the 4 threads constantly updating it. While in the SEQUENTIAL execution the accumulation datastructure is idle while the single thread is parsing the lines to find the next TMAX.

My experiments agree with these expectations and we can see that the SEQUENTIAL execution takes the maximum average running time among these 5 versions of the program.

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.

→ After comparing the temperature averages from each of the program versions, the observation is that the results from all the versions of program execution are almost the same, except from the execution of NO-LOCK program, which contains incorrect temperature averages. The reason being, all 4 threads access the same accumulation data structure concurrently and because there is no lock on this accumulation data structure the updates made by the individual threads are bound to collision and as a result end up with incorrect values.

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.)

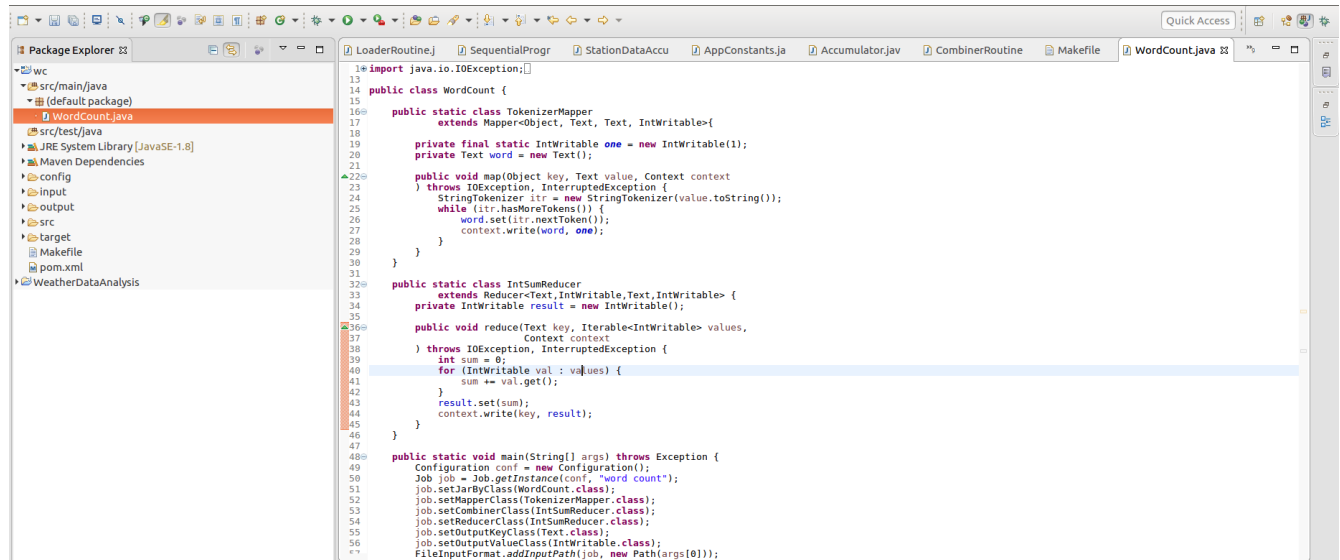
→ As mentioned in the answer for the 2nd question, the running time of SEQUENTIAL execution of the program is slower than the COARSE-LOCK execution of the program. Since the accumulation data structure is kept busy by at least one of the threads during the COARSE-LOCK execution and this is not the case in SEQUENTIAL execution. The SEQUENTIAL execution takes up more time. When these programs have to run an additional Fibonacci computation the observed running time average for the SEQUENTIAL version seems faster.

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.

→ Comparing the running time averages of COARSE-LOCK and FINE-LOCK versions of the program we observe that COARSE-LOCK version of the program is slower when compared to FINE-LOCK version. The reason being, in the COARSE-LOCK version every thread locks the complete accumulation data structure while it is updating one of the stationId's temperature sum. The other threads have to wait even though this thread is updating a different stationId. This is solved in FINE-LOCK and the running time average is faster. When the computation time is made expensive by running the Fibonacci(17), we see that COARSE-LOCK program is still slower than the FINE-LOCK program, also the time average difference between these two executions is increased.

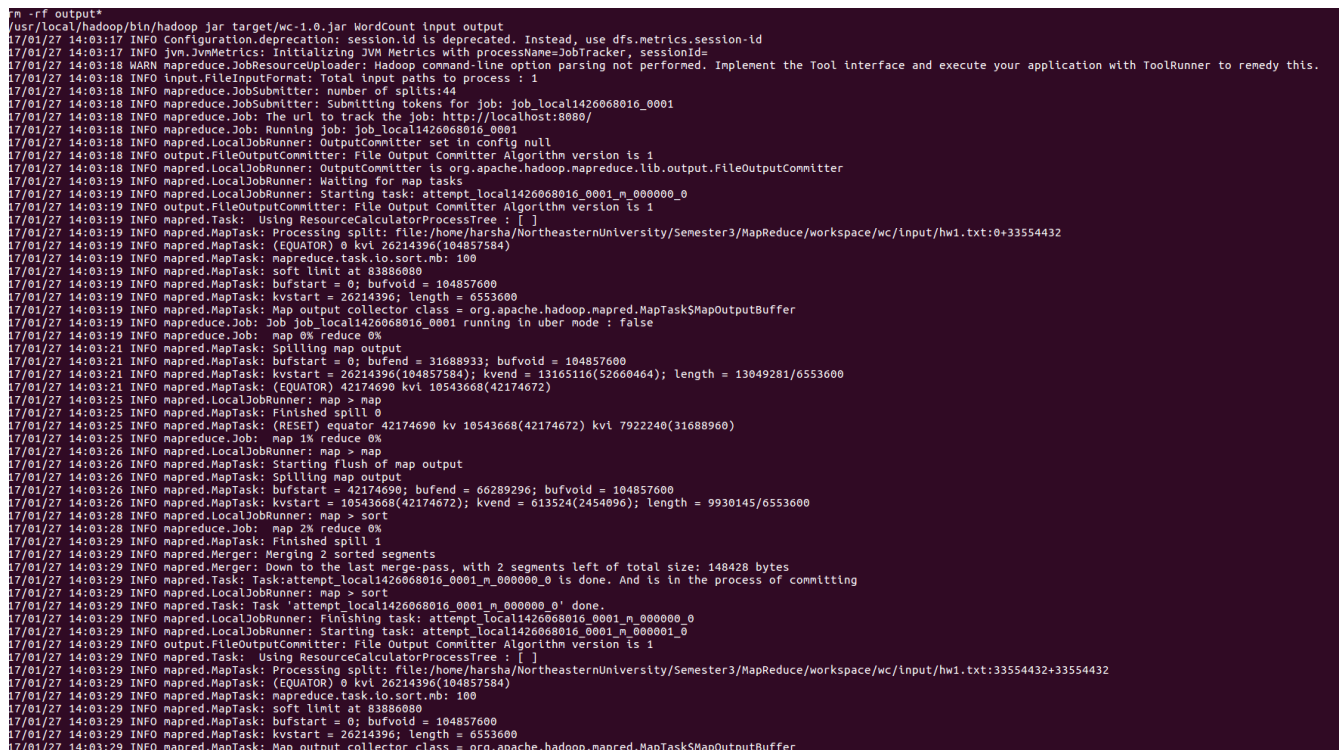
Word Count Local Execution:

The following screenshots show the project directory structure, showing the WordCount.java file in the src directory, also the console output of the successful run of the WordCount with job summary information from Hadoop.



The screenshot shows an IDE with the Package Explorer on the left, displaying the project structure. The main editor shows the WordCount.java file, which contains the following code:

```
1 import java.io.IOException;
13
14 public class WordCount {
15
16     public static class TokenizerMapper
17         extends Mapper<Object, Text, Text, IntWritable>{
18
19         private final static IntWritable one = new IntWritable(1);
20         private Text word = new Text();
21
22     public void map(Object key, Text value, Context context
23         ) throws IOException, InterruptedException {
24         StringTokenizer itr = new StringTokenizer(value.toString());
25         while (itr.hasMoreTokens()) {
26             word.set(itr.nextToken());
27             context.write(word, one);
28         }
29     }
30
31     public static class IntSumReducer
32         extends Reducer<Text, IntWritable, Text, IntWritable> {
33         private IntWritable result = new IntWritable();
34
35     public void reduce(Text key, Iterable<IntWritable> values,
36         Context context
37         ) throws IOException, InterruptedException {
38         int sum = 0;
39         for (IntWritable val : values) {
40             sum += val.get();
41         }
42         result.set(sum);
43         context.write(key, result);
44     }
45
46     public static void main(String[] args) throws Exception {
47         Configuration conf = new Configuration();
48         Job job = Job.getInstance(conf, "word count");
49         job.setJarByClass(WordCount.class);
50         job.setMapperClass(TokenizerMapper.class);
51         job.setCombinerClass(IntSumReducer.class);
52         job.setReducerClass(IntSumReducer.class);
53         job.setOutputKeyClass(Text.class);
54         job.setOutputValueClass(IntWritable.class);
55         FileInputFormat.addInputPath(job, new Path(args[0]));
56     }
57 }
```



The screenshot shows a terminal window with the following output:

```
h -rf output*
/usr/local/hadoop/bin/hadoop jar target/wc-1.0-jar WordCount input output
17/01/27 14:03:17 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/01/27 14:03:17 INFO jvm.Metrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/01/27 14:03:18 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
17/01/27 14:03:18 INFO input.FileInputFormat: Total input paths to process : 1
17/01/27 14:03:18 INFO mapreduce.JobSubmitter: number of splits:44
17/01/27 14:03:18 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1426068016_0001
17/01/27 14:03:18 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/01/27 14:03:18 INFO mapreduce.Job: Job Local1426068016_0001
17/01/27 14:03:18 INFO mapreduce.LocalJobRunner: OutputCommitter set in config null
17/01/27 14:03:18 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/01/27 14:03:18 INFO mapreduce.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
17/01/27 14:03:19 INFO mapreduce.LocalJobRunner: Waiting for map tasks
17/01/27 14:03:19 INFO mapreduce.LocalJobRunner: Starting task: attempt_local1426068016_0001_m_000000_0
17/01/27 14:03:19 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/01/27 14:03:19 INFO mapreduce.Task: Using ResourceCalculatorProcessTree : [ ]
17/01/27 14:03:19 INFO mapreduce.MapTask: Processing split: file:/home/harsha/NortheasternUniversity/Semester3/MapReduce/workspace/wc/input/hw1.txt:0+33554432
17/01/27 14:03:19 INFO mapreduce.MapTask: (EQUATOR) 0 kvl 26214396(104857584)
17/01/27 14:03:19 INFO mapreduce.MapTask: soft limit at 83886080
17/01/27 14:03:19 INFO mapreduce.MapTask: bufstart = 0; bufvoid = 104857600
17/01/27 14:03:19 INFO mapreduce.MapTask: kvstart = 26214396; length = 6553600
17/01/27 14:03:19 INFO mapreduce.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
17/01/27 14:03:19 INFO mapreduce.Job: Job Local1426068016_0001 running in uber mode : false
17/01/27 14:03:21 INFO mapreduce.MapTask: map 0% reduce 0%
17/01/27 14:03:21 INFO mapreduce.MapTask: Spilling map output
17/01/27 14:03:21 INFO mapreduce.MapTask: bufstart = 0; bufend = 31688933; bufvoid = 104857600
17/01/27 14:03:21 INFO mapreduce.MapTask: kvstart = 26214396(104857584); kvend = 13165116(52660464); length = 13049281/6553600
17/01/27 14:03:21 INFO mapreduce.MapTask: (EQUATOR) 42174690 kvl 10543668(42174672)
17/01/27 14:03:25 INFO mapreduce.LocalJobRunner: map > map
17/01/27 14:03:25 INFO mapreduce.MapTask: Finished spill 0
17/01/27 14:03:25 INFO mapreduce.MapTask: (RESET) equator 42174690 kv 10543668(42174672) kvl 7922240(31688960)
17/01/27 14:03:25 INFO mapreduce.Job: map 1% reduce 0%
17/01/27 14:03:26 INFO mapreduce.LocalJobRunner: map > map
17/01/27 14:03:26 INFO mapreduce.MapTask: Starting flush of map output
17/01/27 14:03:26 INFO mapreduce.MapTask: Spilling map output
17/01/27 14:03:26 INFO mapreduce.MapTask: bufstart = 42174690; bufend = 66289296; bufvoid = 104857600
17/01/27 14:03:26 INFO mapreduce.MapTask: kvstart = 10543668(42174672); kvend = 613524(2454096); length = 9930145/6553600
17/01/27 14:03:28 INFO mapreduce.LocalJobRunner: map > sort
17/01/27 14:03:28 INFO mapreduce.MapTask: map 2% reduce 0%
17/01/27 14:03:29 INFO mapreduce.MapTask: Finished spill 1
17/01/27 14:03:29 INFO mapreduce.Merger: Merging 2 sorted segments
17/01/27 14:03:29 INFO mapreduce.Merger: Down to the last merge-pass, with 2 segments left of total size: 148428 bytes
17/01/27 14:03:29 INFO mapreduce.Task: Task: attempt_local1426068016_0001_m_000000_0 is done. And is in the process of committing
17/01/27 14:03:29 INFO mapreduce.LocalJobRunner: map > sort
17/01/27 14:03:29 INFO mapreduce.Task: Task: attempt_local1426068016_0001_m_000000_0 done.
17/01/27 14:03:29 INFO mapreduce.LocalJobRunner: Finishing task: attempt_local1426068016_0001_m_000000_0
17/01/27 14:03:29 INFO mapreduce.LocalJobRunner: Starting task: attempt_local1426068016_0001_m_000000_0
17/01/27 14:03:29 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/01/27 14:03:29 INFO mapreduce.Task: Using ResourceCalculatorProcessTree : [ ]
17/01/27 14:03:29 INFO mapreduce.MapTask: Processing split: file:/home/harsha/NortheasternUniversity/Semester3/MapReduce/workspace/wc/input/hw1.txt:33554432+33554432
17/01/27 14:03:29 INFO mapreduce.MapTask: (EQUATOR) 0 kvl 26214396(104857584)
17/01/27 14:03:29 INFO mapreduce.MapTask: soft limit at 83886080
17/01/27 14:03:29 INFO mapreduce.MapTask: bufstart = 0; bufvoid = 104857600
17/01/27 14:03:29 INFO mapreduce.MapTask: kvstart = 26214396; length = 6553600
17/01/27 14:03:29 INFO mapreduce.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
```

Word Count AWS Execution:

The following screenshots depicts the successful run of the WordCount program on AWS running on three machines (one master node and two workers).

```
aws s3 cp target/wc-1.0.jar s3://harshaj-bucket
upload: target/wc-1.0.jar to s3://harshaj-bucket/wc-1.0.jar
aws s3 rm s3://harshaj-bucket/ --recursive --exclude "*" --include "output*"
aws emr create-cluster \
  --name "WordCount Cluster" \
  --release-label emr-5.2.1 \
  --instance-groups '[{"InstanceCount":2, "InstanceGroupType":"CORE", "InstanceType":"m1.medium"}, {"InstanceCount":1, "InstanceGroupType":"MASTER", "InstanceType":"m1.medium"}]' \
  --applications Name=Hadoop \
  --steps '[{"Args":["WordCount", "s3://harshaj-bucket/input", "s3://harshaj-bucket/output"], "Type":"CUSTOM_JAR", "Jar":"s3://harshaj-bucket/wc-1.0.jar", "ActionOnFailure":"TERMINATE_CLUSTER", "Name":"Custom JAR"}]' \
  --log-uri s3://harshaj-bucket/log \
  --service-role EMR_DefaultRole \
  --ec2-attributes InstanceProfile=EMR_EC2_DefaultRole, SubnetId=subnet-a5b9c7fe \
  --region us-east-1 \
  --enable-debugging \
  --auto-terminate \
  {
    "ClusterId": "j-2KAV6XGHZ69HF"
  }
```

The screenshot shows the Amazon EMR console interface. On the left, there is a navigation menu with options: Amazon EMR, Cluster list, Security configurations, VPC subnets, and Help. The main area displays the details of a cluster named 'WordCount Cluster' with ID 'j-2KAV6XGHZ69HF'. The cluster status is 'Terminated' with the note 'All steps completed'. The creation time is '2017-01-27 15:26 (UTC-5)' and the elapsed time is '22 minutes'. The normalized instance hours are '6'. The cluster summary shows a Master public DNS of 'ec2-54-80-228-4.compute-1.amazonaws.com', Termination protection is 'Off', and Tags are '--'. The hardware section indicates the Master is 'Terminated 1 m1.medium', the Core is 'Terminated 2 m1.medium', and the Task is '--'. The steps section shows two completed steps: 'Custom JAR' and 'Setup Hadoop Debugging'. The bootstrap actions section shows 'No bootstrap actions available'. There are links for 'View cluster details' and 'View monitoring details'.

Name	ID	Status	Creation time (UTC-5)	Elapsed time	Normalized instance hours
WordCount Cluster	j-2KAV6XGHZ69HF	Terminated All steps completed	2017-01-27 15:26 (UTC-5)	22 minutes	6

Summary
Master public DNS: ec2-54-80-228-4.compute-1.amazonaws.com
Termination protection: Off
Tags: --
Hardware
Master: Terminated 1 m1.medium
Core: Terminated 2 m1.medium
Task: --

Steps

Name	Status	Start time (UTC-5)	Elapsed time
Custom JAR	Completed	2017-01-27 15:35 (UTC-5)	11 minutes
Setup Hadoop Debugging	Completed	2017-01-27 15:34 (UTC-5)	2 seconds

Bootstrap Actions

Name
No bootstrap actions available