Report

CS 6240 Sec 02, Assignment 1

Gautam Vashisht

- **File Structure**

  Weather Data Source Code

  Providing two jar file. One for the runs with Fibonacci and one for simple run.

  1. **tempCalc.jar** – It contains all the .java and .class files to compute average temperature, speedup and other calculations for run without Fibonacci.
  2. **tempCalcWithFib.jar** – It contains all the .java and .class file similar to the tempCalc package classes but including the call to Fibonacci as required.
  3. Main method is provided in MainClass.java which will call the Routine Loader class and execute the five versions of programming
  4. Main class is given separately for both the run in their respective folders.
  5. Input file should be provided as run time argument.

- **Weather Data Results**

- Threads used for executing the parallel programming are 5 in each program of multithreading.

For part B –

Runs without Fibonnacci:

******Sequential Run******

Minimum Time for Sequential Run: 2224

Maximum Time for Sequential Run: 5438

Average Time for Sequential Run: 2572.7

****

******No Lock Run******

Minimum Time for NoLock Run: 1024

Maximum Time for NoLock Run: 1087

Average Time for NoLock Run: 1056.1

Speed Up: 2.436038253953224

****

******Coarse Lock Run******

Minimum Time for CoarseLock Run: 1135

Maximum Time for CoarseLock Run: 1197

Average Time for CoarseLock Run: 1151.7

Speed Up: 2.23382825388556

****

******Fine Lock Run******

Minimum Time for FineLock Run:1122

Maximum Time for FineLock Run:1169

Average Time for FineLock Run:1143.6

Speed Up: 2.249650227352221

****

******No Sharing Run******

Minimum Time for NoSharing Run:1140

Maximum Time for NoSharing Run:1180

Average Time for NoSharing Run:1156.8

Speed Up: 2.299100968188105

For C–

Runs with Fibonnacci:

******Sequential Run******

Minimum Time for Sequential Run: 2244

Maximum Time for Sequential Run: 5469

Average Time for Sequential Run: 2621.3

****

******No Lock Run******

Minimum Time for NoLock Run: 1060

Maximum Time for NoLock Run: 1123

Average Time for NoLock Run: 1080.6

Speed Up: 2.4257819729779757

****

******Coarse Lock Run******

Minimum Time for CoarseLock Run: 1163

Maximum Time for CoarseLock Run: 1203

Average Time for CoarseLock Run: 1183.5

Speed Up: 2.214871144909168

****

******Fine Lock Run******

Minimum Time for FineLock Run: 1150

Maximum Time for FineLock Run: 1204

Average Time for FineLock Run: 1172.8

Speed Up: 2.2350784447476126

****

******No Sharing Run******

Minimum Time for NoSharing Run: 1088

Maximum Time for NoSharing Run: 1158

Average Time for NoSharing Run: 1122.9

Speed Up: 2.3537269569863746

1. As per understanding, NO-LOCK code should finish fastest as it is processing the data parallel using 5 threads. Also, there is no lock or synchronization to delay the execution of program. Though, it might have data inconsistency but it will finish fastest.
   Results are as per expected. In part b and c, its average time is lowest only.
   1056.1 ms for B
   1080.6 ms for C

2. Sequential Program should finish slowest, as it is processing the data in sequence using single thread main. Result is as per expected –
   2572.7 ms for B
   2621.3 ms for C

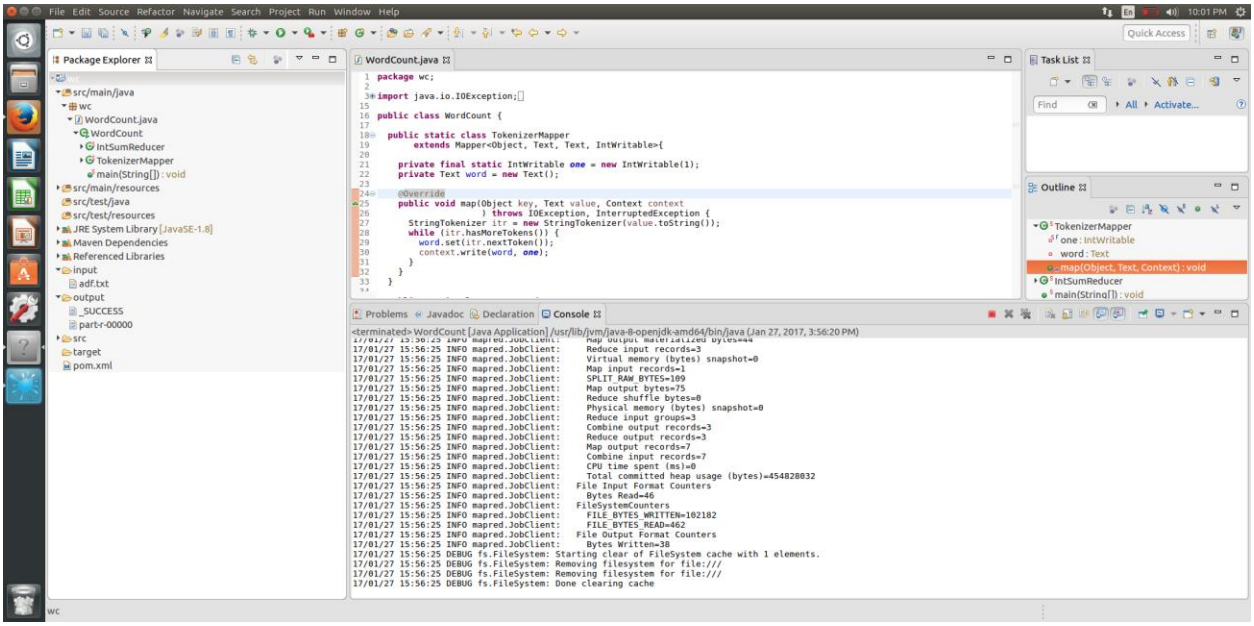
3. There is mismatch for the average of temperatures given by NO-LOCK. It is as expected due to threads without any lock or synchronization.
   For few executions, NO-LOCK is throwing an exception getting generated from put method of hash map –
   Exception in thread "t2" java.lang.ClassCastException: java.util.HashMap$Node cannot be cast to java.util.HashMap$TreeNode.

4. For b) sequential run is taking 2572.7 ms while 1151.7 ms for the coarse lock run.
   For c) sequential run is taking 2621.3 ms while 1183.5 ms for the coarse lock run.
   As per me, Sequential run is slower than coarse lock as coarse lock is multithreaded programming. Though, there is lock on data structure and other threads have to wait but still other part of the code apart from getting and putting into map will act parallel which will make it faster than Sequential Run.

5. Difference between them decreases as I think running extra computation will cost Coarse Lock run more as now each thread has to wait for Fibonacci call also while Fine lock run will access the Fibonacci call parallel.

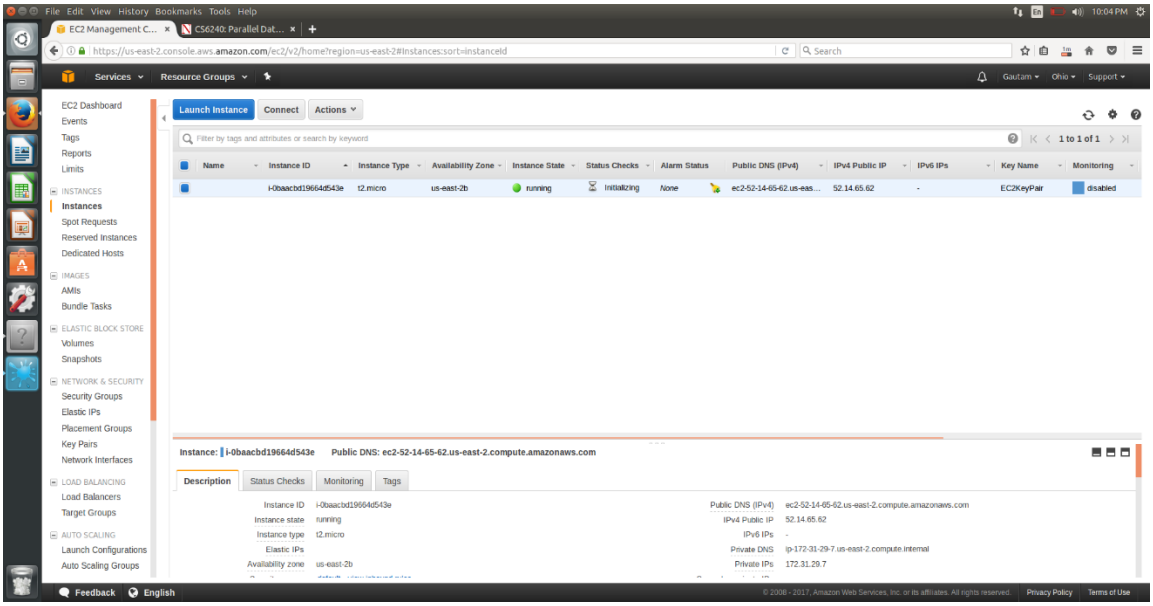
Word Count Local Execution

- Snapshot for IDE Eclipse for WordCount.java

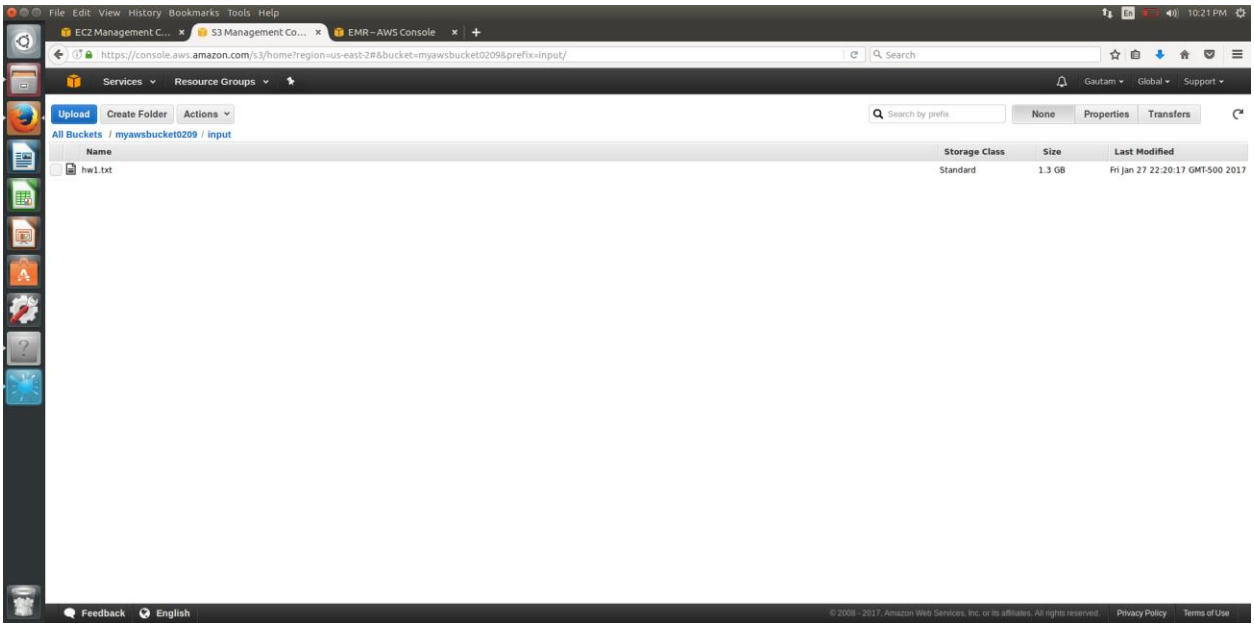


Word Count AWS Execution

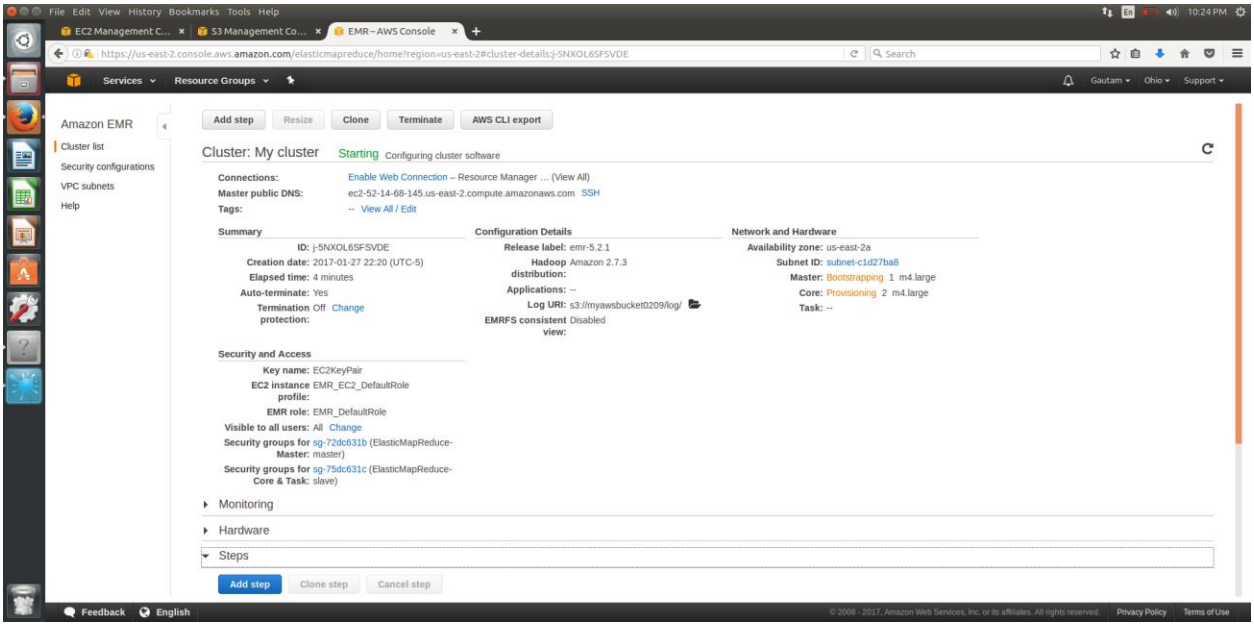
EC2 Screenshot



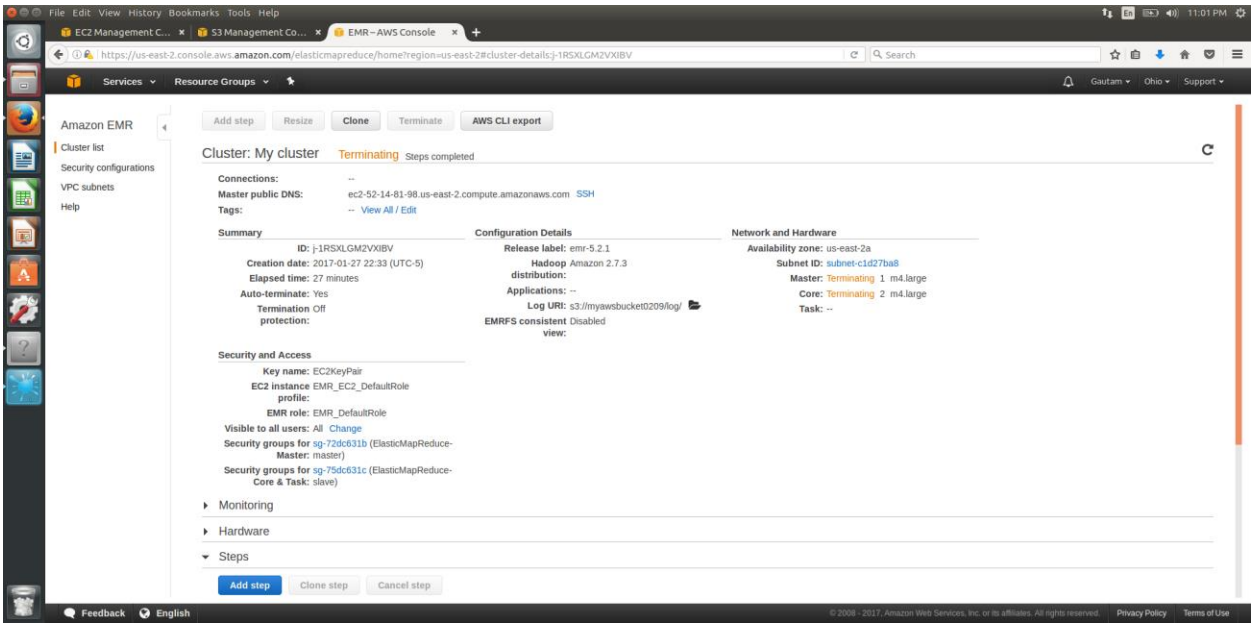
S3 Bucket with I/P data



Cluster



Cluster Completed



Output File