Sravani Beeram
CS6240 Parallel Data Processing using Mapreduce
Homework 1

Running Time without Fibonacci delay

Program	Minimum Time(ms)	Maximum Time(ms)	Average Time(ms)
No Lock	417	512	446
Fine Lock	429	530	452
No Sharing	409	639	477
Coarse Lock	766	804	784
Sequential	755	1901	885

Running Time with Fibonacci delay

Program	Minimum Time(ms)	Maximum Time(ms)	Average Time(ms)
No Lock	439	689	524
Fine Lock	427	770	610
No Sharing	444	766	658
Coarse Lock	777	1041	901
Sequential	785	2377	1134

Number of Threads used for running the program : 2

Speed up calculation for Fibonacci delay :

No Lock :2.16
Fine Lock:1.86
No Sharing:1.72
Coarse Lock:1.26

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.

No-Lock is expected to finish fast as data is shared between the threads and threads will act independently to update the data structure.Run times of my programs confirm the same

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.*

Sequential is expected to finish slowest as there is only one thread in the system to complete the execution.Hence it will be slower than the multithreaded programs. My programs results confirm that SEQ does take maximum time for execution for case B.

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.

Except No_Lock program, temperature averages returned by the other program version were same.Sometimes No_Lock program returned Null Pointer Exception .This was because as one thread had already access to object ID but had not updated in the common data structure.
And second thread already tries to access the Id and doesn't get any value from the data structure.

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.)

In general Sequential should be slowest without Fibonacci and Coarse-lock slowest with Fibonacci. Sequential has only one thread to execute the program and is slower compared to multiple thread programs.Hence,without Fibonacci ,Sequential is slowest and program run times confirm the same.

With Fibonacci, Coarse lock has a thread that locks the object and no other threads have access to the object and fibonacci slows it down further. My coarse-lock code with fibonacci is

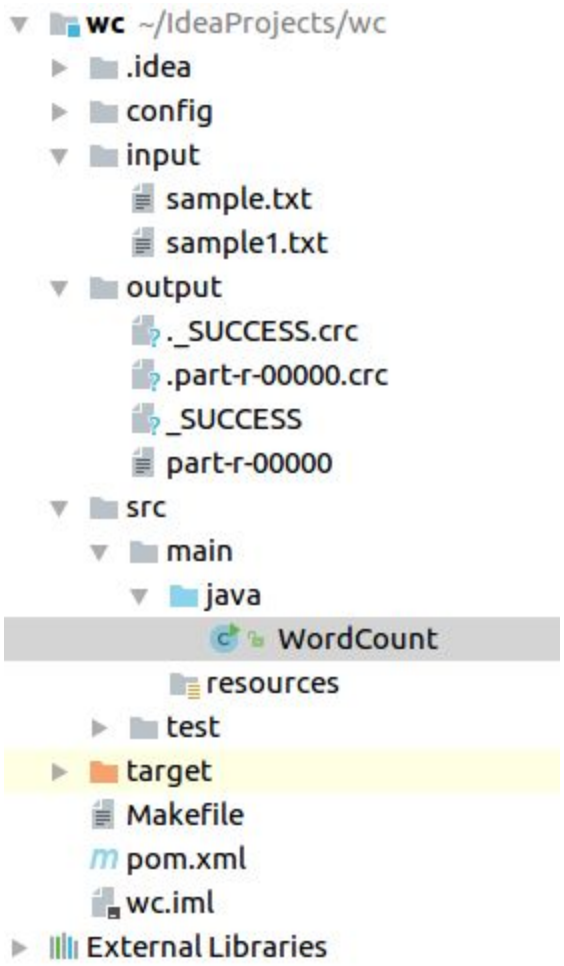
60% slower than sequential but other times sequential is slower.It could be because the lock on the object is release quikly and hence runs faster than sequential.It would have been more evident if the task was bigger.

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.

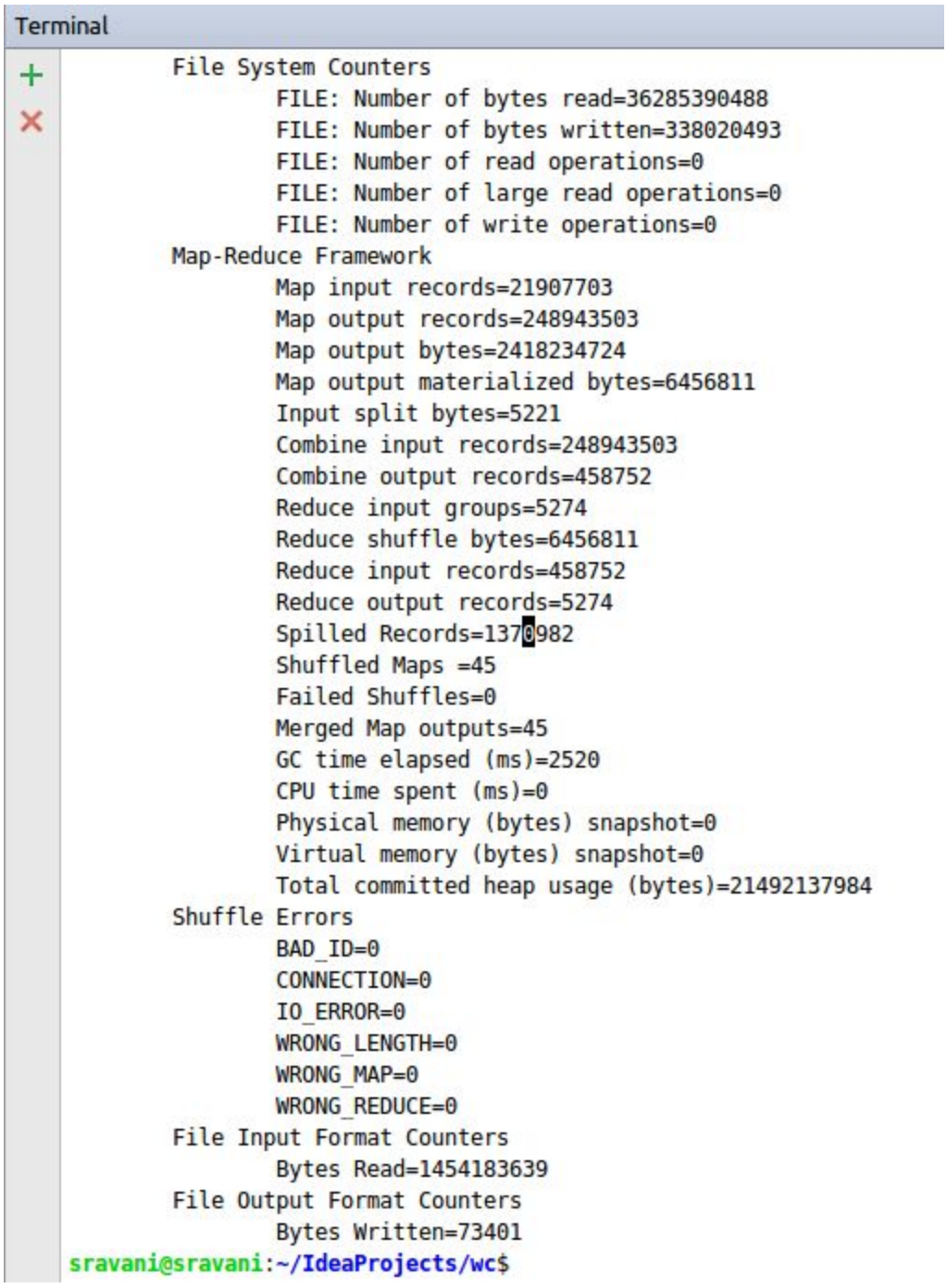
Higher computation cost doesn't effect difference between Coarse-lock and Fine-lock since in both of them before we update the values in accumulated data structure we make the threads wait.Code results depict the same.

Word Count Local Execution

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



2.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.



Word Count AWS Execution

1.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