

Notice : All computations have been performed on i7 processor with quad cores, hence at any given time 4 threads should be running simultaneously.

Answering questions asked as a part of Assignment 1 -

Without Fibonacci →

Program	Average Time	Min Time	Max Time
Coarse Lock	1001.9	955	1075
Sequential Lock	954.6	694	1865
No Shared Lock	378.6	362.0	439
Fine Lock	365	345	419
No Lock	356.4	340	397

With Fibonacci →

Program	Average Time	Min Time	Max Time
Sequential Lock	981.3	681	1561
Coarse Lock	844	887	966
No Shared Lock	391	367	444
Fine Lock	370.4	350	464
No Lock	363.8	335	404

---

Speed Up for the following locks are →

Coarse Lock = 1.163

No Shared Lock = 2.509

Fine Lock = 2.65

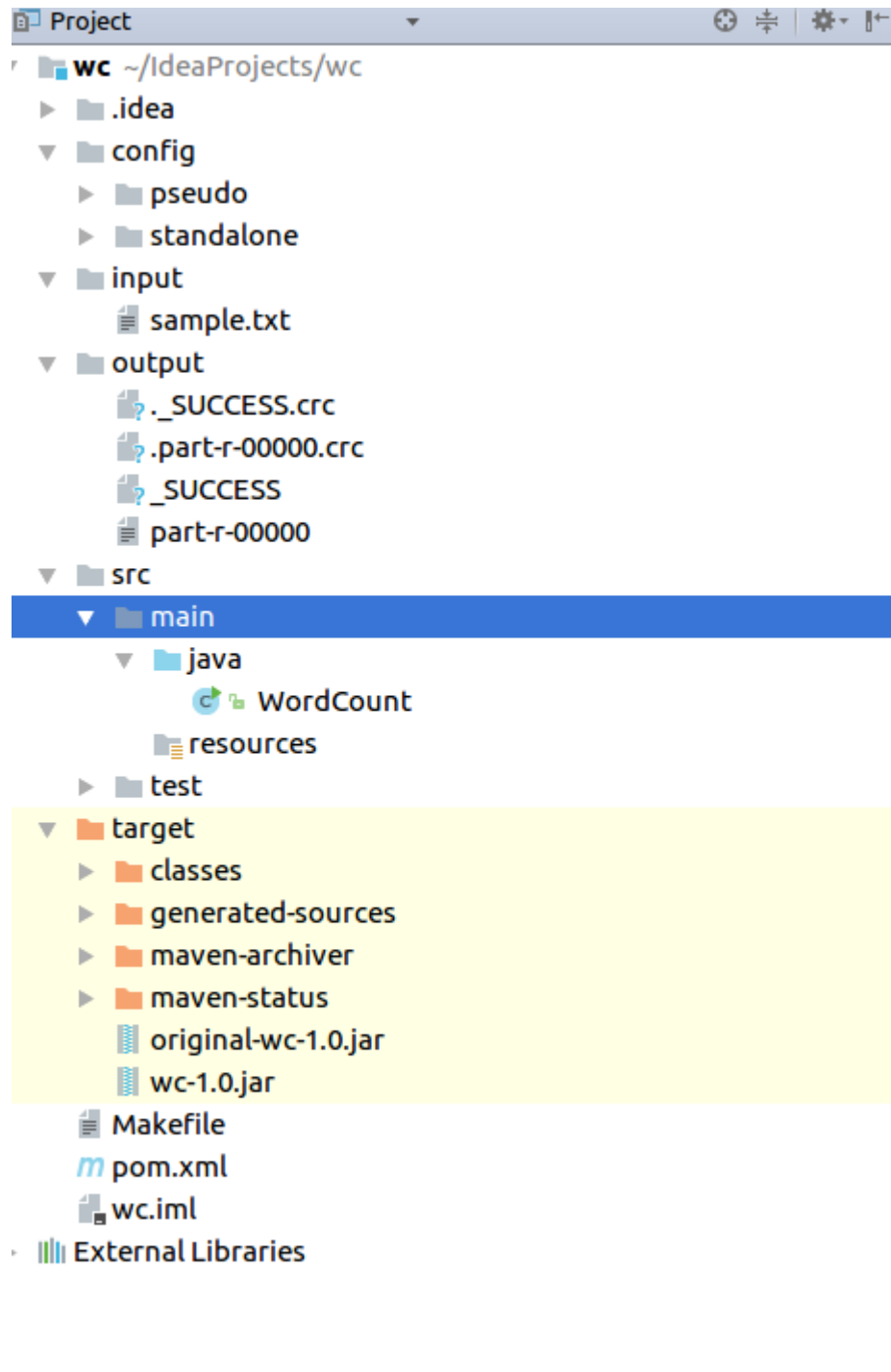
No Lock = 2.698

- 
1. Which program would you expect to complete fastest and why? Do experiments confirm the results ? Explain with reasons
  - A. My hypothesis for the same is, if any give set of threads are allowed to run with all resources and without any locks/holds, the threads will start wait and stop as and when it is convenient for them. When the processor feels a thread is taking lot of time, it gives priority to another one, hence effectively reducing time required for completion of code. The same principle is applied when we create NO\_LOCK, hence it should be the fastest.

I have run my program several times and observed NO-LOCK code is running the fastest which confirms my hypothesis.

2. Which program would you normally expect to complete the slowest and why? Do experiments confirm your your observation?
    - A. I expect SEQUENTIAL to complete the slowest because it is like there is only one thread in the system and entire execution is supposed to be completed by it. This will definitely be slower than multiple threads working on the same problem to find average of max temperature. The results from the program assure the same observation, SEQ does take max time for execution(only for case B).
  3. Compare temperatures, report if anyone is incorrect or if programs crashed because of concurrent processes.
    - A. Temperatures remain the same for all scenarios except NO\_LOCK which is expected since multiple threads access and make changes to objects without proper completion of process/snippet.  
I also faced crash while running the code in NO\_LOCK i.e. NullPointerException which was because, my thread one had already access to object(ID) but had not updated in the accumulated data structure values(max temp and count), but second thread went ahead, accessed the ID but did not get any value in the data structure which cause the Exception  
/\*\*Add temperatures here \*/
  4. Compare run times of SEQ and COARSE\_LOCK. Why is one slower than other?
    - A. Logically speaking, SEQUENTIAL should be slowest *without Fibonacci* and COARSE\_LOCK slowest *with Fibonacci*.  
The execution of sequential is like there is only one thread in the system and it executes program from start to end. This is definitely slower than multiple threads running the same code. Hence, without Fibonacci we have Sequential as slowest, which is observed through results as well.  
When we have Fibonacci, in a coarse lock, it so happens that a thread locks the object, hence no other threads have access to the object. In addition to this, Fibonacci slows the execution even more. This in turn according to me may become slower than sequential.  
My COARSE\_LOCK code with Fibonacci 3/5 times is slower than sequential but other times sequential is slower. This could be possible because the lock on the objects is released quickly and hence runs faster than sequential, but I feel if the programming task is bigger, COARSE\_LOCK would end up being slower than SEQUENTIAL.
  5. How does higher cost computation affect difference between COARSE\_LOCK and FINE\_LOCK?
    - A. The computation remains the same for both COARSE\_LOCK and FINE\_LOCK. Fibonacci does not affect computation of COARSE\_LOCK and FINE\_LOCK since both ways before we update the values in accumulated data structure we ask the threads to wait. There is no change in the ratio of execution times observed.  
The same is observed while running the code.
-

Standalone Screen shot →



# CS6240 Assignment 1 Section 1

Priyanka Shenoy

```
wc - ~/IdeaProjects/wc - [wc] - ~/IdeaProjects/wc/Makefile - IntelliJ IDEA 2016.3.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
wc | Makefile
Project
core-site.xml | wc | Makefile | WordCount.java |
Terminal
17/01/28 20:03:16 INFO reduce.MergeManagerImpl: Merging 1 files, 6456537 bytes from disk
17/01/28 20:03:16 INFO reduce.MergeManagerImpl: Merging 0 segments, 0 bytes from memory into reduce
17/01/28 20:03:16 INFO mapred.Merger: Merging 1 sorted segments
17/01/28 20:03:16 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 6456528 bytes
17/01/28 20:03:16 INFO mapred.LocalJobRunner: 44 / 44 copied.
17/01/28 20:03:16 INFO Configuration.deprecation: mapred.skip.on is deprecated. Instead, use mapreduce.job.skiprecords
17/01/28 20:03:16 INFO mapred.Task: Task:attempt_local1899727981_0001_r_000000_0 is done. And is in the process of committing
17/01/28 20:03:16 INFO mapred.LocalJobRunner: 44 / 44 copied.
17/01/28 20:03:16 INFO mapred.Task: Task:attempt_local1899727981_0001_r_000000_0 is allowed to commit now
17/01/28 20:03:16 INFO output.FileOutputCommitter: Saved output of task 'attempt_local1899727981_0001_r_000000_0' to file:/home/ps/IdeaProjects/wc/output/_temporary/0/task_local1899727981_0001_r_000000
17/01/28 20:03:16 INFO mapred.Task: reduce > reduce
17/01/28 20:03:16 INFO mapred.Task: Task 'attempt_local1899727981_0001_r_000000_0' done.
17/01/28 20:03:16 INFO mapred.LocalJobRunner: Finishing task: attempt_local1899727981_0001_r_000000_0
17/01/28 20:03:16 INFO mapred.LocalJobRunner: reduce task executor complete.
17/01/28 20:03:17 INFO mapreduce.Job: map 100% reduce 100%
17/01/28 20:03:17 INFO mapreduce.Job: Job job_local1899727981_0001 completed successfully
17/01/28 20:03:17 INFO mapreduce.Job: Counters: 30
File System Counters
  FILE: Number of bytes read=34824536498
  FILE: Number of bytes written=326049668
  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=21907700
  Map output records=248943500
  Map output bytes=2418234700
  Map output materialized bytes=6456795
  Input split bytes=4884
  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)=2476
  Total committed heap usage (bytes)=20609236992
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
ps@ps:~/IdeaProjects/wc$
```

Services

Resource Groups

Priyanka Shenoy

Oregon

Support

Amazon EMR

Cluster list

Security configurations

VPC subnets

Help

Add step

Resize

Clone

Terminate

AWS CLI export

Cluster: WordCount Cluster

Terminated

Steps completed

Connections:

Master public DNS:

Tags:

--

ec2-54-202-123-53.us-west-2.compute.amazonaws.com

--

SSH

Summary

Configuration Details

Network and Hardware

ID: j-KDJDEPR3YJ9T

Creation date: 2017-01-28 23:28 (UTC-5)

End date: 2017-01-28 23:50 (UTC-5)

Elapsed time: 21 minutes

Auto-terminate: Yes

Termination protection: Off

Release label: emr-5.2.1

Hadoop distribution: Amazon 2.7.3

Applications: --

Log URI: s3://mr-ps/logs/

EMRFS consistent view: Disabled

Availability zone: us-west-2a

Subnet ID: subnet-54ef4a33

Master: Terminated 1 m1.medium

Core: Terminated 2 m1.medium

Task: --

Security and Access

Key name: --

EC2 instance profile: EMR\_EC2\_DefaultRole

EMR role: EMR\_DefaultRole

Visible to all users: All

Security groups for sg-76b83b0e (ElasticMapReduce-Master: master)

Security groups for sg-77b83b0f (ElasticMapReduce-Core & Task: slave)

Monitoring

Hardware

Steps

Configurations

Events

Bootstrap Actions