Class Number: SEC 01(BOS-2-TR) (CRN: 13036)

HW Number: 1 **Name: Anoop Pai**

Part B (Without Fibonacci):

Maximum time observed for sequential processing: 3317

Minimum time observed for sequential processing: 2170

Average time observed for sequential processing: 2337

Maximum time observed for concurrent processing with no locks: 740

Minimum time observed for concurrent processing with no locks: 588

Average time observed for concurrent processing with no locks: 652

Maximum time observed for concurrent processing with coarse locks: 810

Minimum time observed for concurrent processing with coarse locks: 687

Average time observed for concurrent processing with coarse locks: 720

Maximum time observed for concurrent processing with fine locks: 702

Minimum time observed for concurrent processing with fine locks: 602

Average time observed for concurrent processing with fine locks: 666

Maximum time observed for concurrent processing with no sharing data structure: 755

Minimum time observed for concurrent processing with no sharing data structure: 591

Average time observed for concurrent processing with no sharing data structure: 653

Part C (With Fibonacci):

Maximum time observed for sequential processing: 11520

Minimum time observed for sequential processing: 7704

Average time observed for sequential processing: 8161

Maximum time observed for concurrent processing with no locks: 2442

Minimum time observed for concurrent processing with no locks: 2131

Average time observed for concurrent processing with no locks: 2161

Maximum time observed for concurrent processing with coarse locks: 2859

Minimum time observed for concurrent processing with coarse locks: 2409

Average time observed for concurrent processing with coarse locks: 2517

Maximum time observed for concurrent processing with fine locks: 2422

Minimum time observed for concurrent processing with fine locks: 2108

Average time observed for concurrent processing with fine locks: 2281

Maximum time observed for concurrent processing with no sharing data structure: 2469

Minimum time observed for concurrent processing with no sharing data structure: 2069

Average time observed for concurrent processing with no sharing data structure: 2242

There were four worker threads used in the multithreaded version of the programs.

Find the speedup for the multithreaded programs as below:

Part B:

NO-LOCK: 3.584

COARSE-LOCK: 3.24

FINE-LOCK: 3.50

NO-SHARING: 3.578

Part C:

NO-LOCK: 3.77

COARSE-LOCK: 3.24

FINE-LOCK: 3.577

NO-SHARING: 3.64

1) Before I started on the assignment, I expected the program with no locks (NO-LOCK) to have the fastest running time and therefore the best speedup. My experiment only reinforces my thoughts. In the case of no-sharing architecture, there is an aggregation step that has to be performed after the four threads have ended execution. Meanwhile, there is no such overhead for no-locks as updates are made to a global data structure. However, it has to be noted that having no locks means that updates made by one thread can be lost by another thread which results in incorrect results. The other versions consist locks on either the value to be updated or the data structure itself which affects the running time of the program as a thread will have to wait on a resource that is already occupied by another thread.

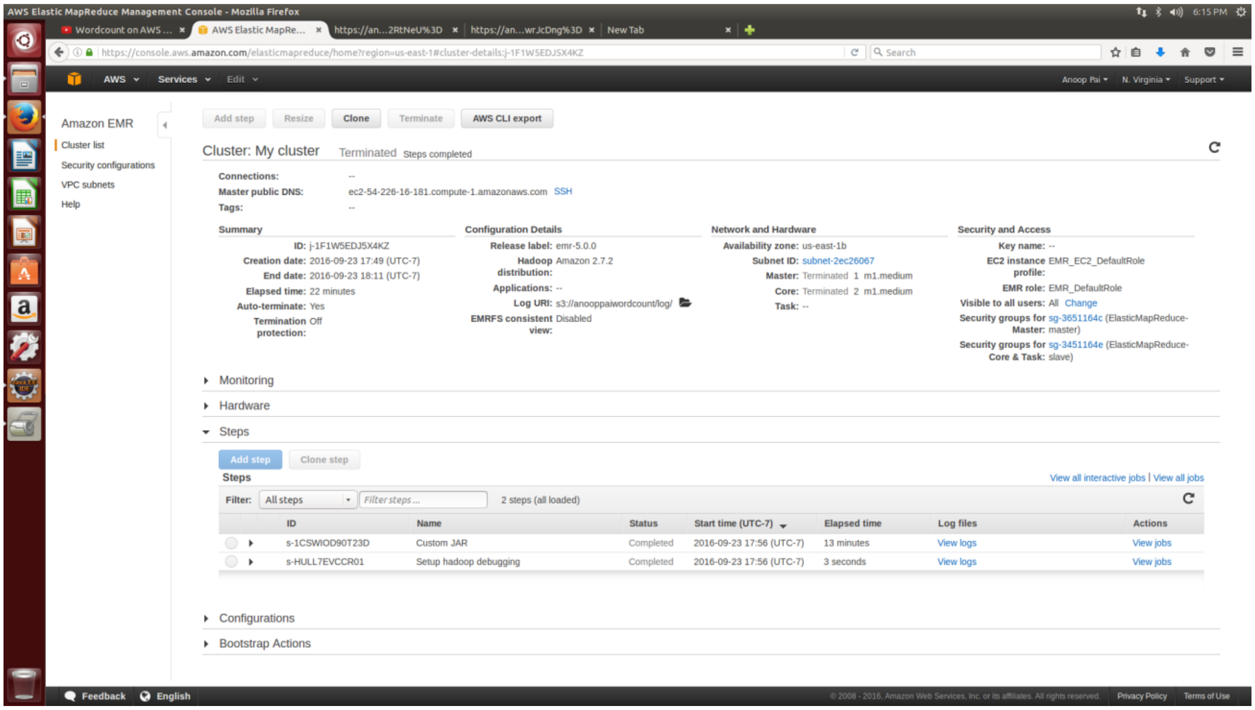
2) Without much doubt, the sequential version (SEQ) of the program was expected to finish the slowest. The experiments clearly reflect the same. In the case of all other versions, threading is involved which drastically improves its running time by having several (four in this case) processes running concurrently. Although there is an aggregate step in NO-SHARING, it would still run faster than the sequential version as it would have four threads running in parallel.

3) After comparing the output of all five versions of the program, it is clear that only the concurrent version with no-locks outputs incorrect results. As mentioned above, as there is no lock present at the data structure level or at the value that is being updated, a thread can easily be making changes to a variable that is lost by another thread which is accessing the same variable at the same time. This absence of any form of synchronization between threads results in incorrect output.

4) As observed in the values of both Part B and Part C above, the running time of COARSE-LOCK is faster than SEQ. This can be explained by the fact that, despite having a coarse lock on the data structure, there are still four worker processes running at the same time. Therefore, although process A is waiting for process B to release a lock so that Process A can acquire a resource that Process B is making changes to, the rest of computation (besides any update to that resource) can be performed by Process A while the same cannot be said about SEQ version of the program.

5) As shown in the speed up comparison of the different versions, higher computation cost in Part C increases the gap between COARSE-LOCK and FINE-LOCK than the difference seen in Part-B, which means the FINE-LOCK runs faster in Part C than in Part B. Although this is not supported by my experiments, I believe that FINE-LOCK should've run slower in Part C than in Part B relative to COARSE-LOCK as extra computational cost should extend the time that a process holds on to a lock and therefore decreases its running time.

AWS Word Count execution:



Word Count Local execution: