CS 6240 Parallel Data Processing in MapReduce - 02
Homework 1
Dheeraj Joshi

Weather Data Results

Running Times & Speedup:

No. of Threads: 4

1. Executed with Fibonacci(17) Delay on updates:

Version 1: Sequential

Average	3104.7
Min	2586
Max	7035

Version 2: No Lock

Average	2577.2
Min	2536
Max	2605
Speedup	1.204679497

Version 3: Coarse Lock

Average	2625.5
Min	2581
Max	2668
Speedup	1.182517616

Version 4: Fine Lock

Average	2729.8
Min	2639
Max	2792
Speedup	1.137336069

Version 5: No Sharing

Average	2725.9
Min	2679
Max	2800
Speedup	1.138963278

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 1
Dheeraj Joshi

2. Executed without Fibonacci(17) Delay on updates:

Version 1: Sequential

Average	3151.6
Min	2533
Max	8341

Version 2: No Lock

Average	2558.5
Min	2516
Max	2596
Speedup	1.231815517

Version 3: Coarse Lock

Average	2622.2
Min	2563
Max	2690
Speedup	1.201891541

Version 4: Fine Lock

Average	2720
Min	2654
Max	2771
Speedup	1.158676471

Version 5: No Sharing

Average	2697.1
Min	2657
Max	2752
Speedup	1.16851433

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 1
Dheeraj Joshi

Questionnaire:

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.

Ans: I expect the NO-LOCK version to be the fastest normally as the job of accumulation of sums and counts is distributed across multiple threads. Also, having no lock on the shared data structure eliminates any chances of experiencing concurrent read/write access on values and might produce inconsistent values. The experiments conducted exhibit this behavior as depicted by the statistics above.

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

Ans: I expect the SEQUENTIAL version to be the slowest normally as all computations for accumulation of sums and counts are done sequentially. The experiments conducted also exhibit this behavior as depicted by the statistics above.

3.Compare the temperature averages returned by each program version. Report if any of them is **incorrect**.

Ans: The average temperatures returned for SEQ, COARSE-LOCK, FINE-LOCK and NO-SHARING versions appear consistent and their approach confirms the same. The average temperatures for NO-LOCK version do tend to change when number of threads are increased in order to create concurrent read/write access by threads. These threads do not know about the current status of the shared data structure at any given point in time and therefore, do not produce consistent values.

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 1
Dheeraj Joshi

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.)

Ans: The running times for SEQ version are the highest among all the versions due to the absence of any parallel processing functions. On the other hand, running times for COARSE-LOCK are lower than the SEQ version due to presence of parallel computations for accumulating sums and counts of subsets of data in a shared data structure. The COARSE-LOCK does have an additional overhead of locking the entire data structure when an another thread is writing a value to it. Considering both B and C scenarios, the average running times are consistent for both the versions but the speedup reduces marginally for COARSE-LOCK based as exhibited in the above stats. (Observation: On increasing the number of threads from 4, the average running times for both versions head to convergence).

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.

Ans: The higher computation cost in part C introduces increase in runtime for a program version that waits before updating the shared data structure too often. This means that the FINE-LOCK version does have a substantial decrease in average running time as compared to COARSE-LOCK due to its granular locking mechanism. The waiting period for COARSE LOCK is more and therefore, takes more time to execute. This is not exactly exhibited in the statistics shown above despite of their proximity in value but when experimented with multiple runs and increased number of threads, this is evident.