Parallel Data Processing with Map Reduce

CS 6240

HomeWork-1

Satya Emani

Weather data results (version B):

Number of worker threads: 3

No Fibonacci

Thread Version	Speed up	Minimum(millis)	Maximum(millis)	Average(millis)
Sequential		888	4599	1287.0
No locks	2.65	445	544	485.0
Coarse Lock	2.10	585	705	611.0
Fine Lock	2.37	506	672	541.0
No Sharing	2.32	604	488	554.0

Speedup: Parallel (average)time / sequential (average)time.

<u>Note:</u> The extreme difference in minimum and maximum values could be due to the JIT compiler trying to access the classes first time and later not needing to get them.

Weather data results (version C):

Number of worker threads: 3

With Fibonacci(17)

Thread Version	Speed up	Minimum(millis)	Maximum(millis)	Average(millis)
Sequential		11218	15780	11813.0
No locks	1.45	4199	10298	8143.0
Coarse Lock	0.69	12719	20961	17015.0
Fine Lock	1.29	7523	10713	9088.0
No Sharing	1.30	7558	11236	9074.0

Q1:

I expected "No locks" to be fastest because there is no wait time for any thread, so multiple threads can concurrently access and modify data, however, there is a possibility of program getting crashed with no locks so we cannot consider it.

Next I expected "Fine lock" to be fastest as it locks the very minimal portion of the data structure but according to the experiment I found that "no sharing" was faster so may be in this case due to less number of computations in aggregation it's faster.

Q2:

"Sequential" is expected to be slowest of all as there is no parallel access at all. It is a single thread processing data one by one. In the table we can see the average time for sequential is highest.

Q3:

All the temperature averages are consistent except for "No Locks" as the concurrent modification leads to inconsistent data. Moreover, No locks also crashed several times throwing exceptions(possible when a thread tries to access corrupted data structure) and not calculating the remaining Tmax values of station id's.

Q4:

Version B: In this case Sequential is slower than Coarse lock as there is absolutely no parallelism in sequential.

Version C: In this case Coarse is more than sequential because the lock is holding the entire data structure for as long as another similar thread processing the data, which means coarse is almost acting as sequential. The values in the table could be mostly due the token exchange time.

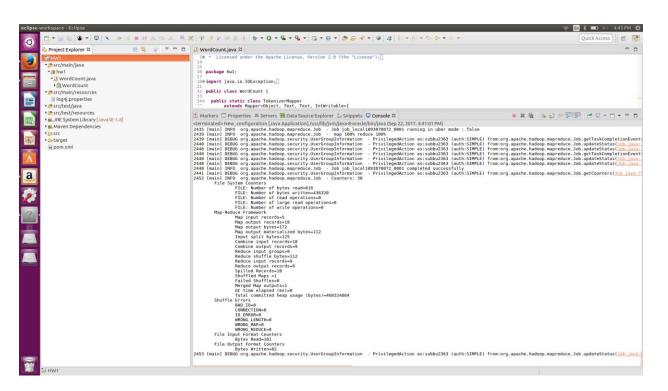
Q5:

The difference between "coarse lock" and "fine lock" was not much in version B. However, in version C the lock is placed on the entire data structure for a very long time making the threads wait unnecessarily (even if they want to access different tuples), but in "fine lock" the lock is placed at a granular level allowing other threads to access other tuples.

Huge wait time for the whole data structure increased the difference between times of coarse lock and fine lock.



1 master, 2 workers



Last 20 lines of the output

Word count program directory.