CS5300
PROG ASGN 3

Name : J SAI NISHITH
Roll No. :CS18BTECH11018

Wait-Free Atomic Snapshot Implementations : Comparing the Solutions of MRSW and MRMW

Goal: The goal of this assignment is to implement the two Atomic Wait-Free Snapshots algorithms.

Comparison of the performance of average time and worst-case taken for the MRSW and MRMW snapshot implementations.

CODE EXPLANATION:
The code for mrsw snapshot has made use of **std::atomic** for getting atomic MRSW registers.
There are ‘n’ writer threads which update the snapshot with a time delay that is exponentially distributed with an average of μw microseconds. and 1 thread which collects snapshots for k times with a time delay that is exponentially distributed with an average of μs microseconds.

For mrsw snapshot, i have used arrays of StampedSnaps (contain stamp, value, snapshot size,snapshot) a_table.
Whenever a writer updates a value in the register, we update the stamp and store the new value and also store a snapshot by invoking scan( ) method.
Whenever a thread invokes the scan( ) method, if we obtain a clean double collect, we return it. Else we return the snapshot the thread which has updated twice when the reader thread was trying to get the snapshot. Bool moved array is used to check if a thread has updated twice.

This method doesn't work for mrmw because this method assumes that a particular element of snapshot is being updated by a particular thread.

We extend the similar idea to MRMW. We store pid along with the stamp and value of each register to keep the track of which thread has updated it. He use helparray which is MRSW register and this is used by writer threads to store snapshot.
We maintain a vector to check if a particular thread has visited twice. If it does, we return it's snapshot.

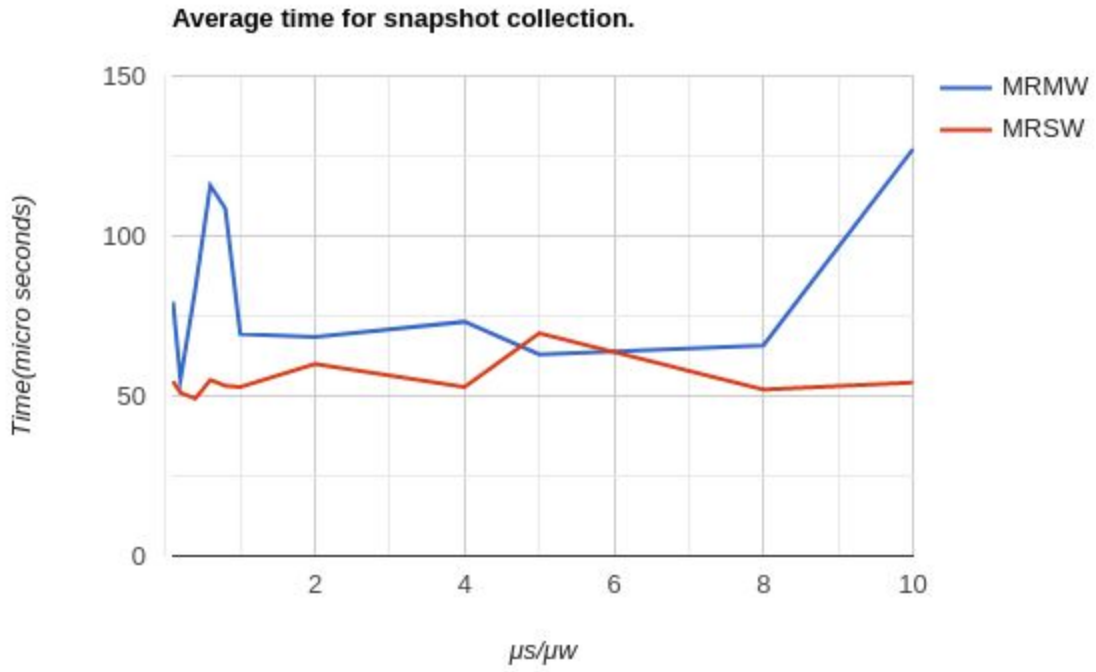
For following observations i took n as 10, M as 20 and k as 5 as mentioned.

GRAPHS

Graph 1:
The y-axis of the graph is the average time taken for the snapshot algorithm to complete.

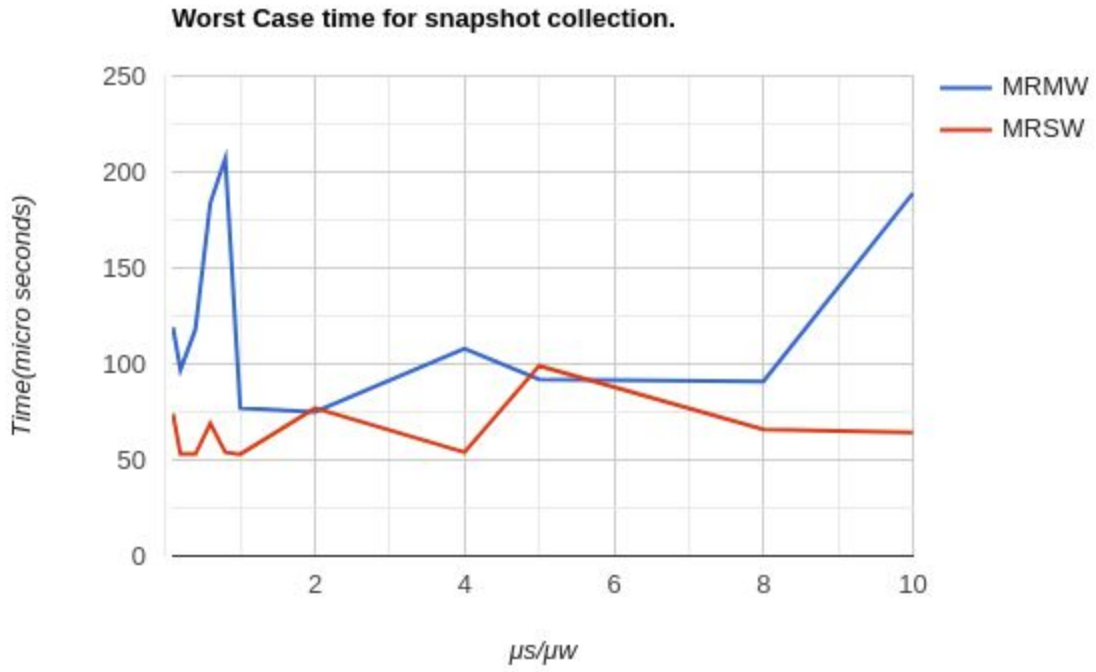
The x-axis of the graph will consist of a ratio of μs/μw, i.e. the ratio of the average delays of the snapshot thread to the writer thread. It will specifically consist of the following 11 values:
0.1 0.2 0.4 0.6 0.8 1 2 4 5 8 10

Graph(in the next page)



Graph 2:
The y-axis of the graph is the worst case time taken for the snapshot algorithm to complete.

The x-axis of the graph will consist of a ratio of μs/μw, i.e. the ratio of the average delays of the snapshot thread to the writer thread. It will specifically consist of the following 11 values:
0.1 0.2 0.4 0.6 0.8 1 2 4 5 8 10



OBSERVATIONS:

MRSW is faster than MRMW in some cases and in some cases MRSW overshoots MRSW. So there is no clear pattern.

For MRSW : Every scan() or update() returns after at most O(n^2) reads or Writes.

For MRMW : Every scan() or update() returns after at most O(n*m) reads or writes.

The best case complexity O(n) for mrsw and O(m) for mrmw.(which is the same because n=m for mrsw)
The worst case complexity O(n^2) for mrsw and O(n*m) for mrmw.

For our current comparison

For MRSW n=10 and also m=10
For MRMW n=10 and m= 20
So that could be the reason(higher value of m) because of which mrmw has taken more time for some part but not completely. But mrsw is more in some cases.

The best case and worst case complexities are the same. So it's very tough to conclude a faster one out of MRSW and MRMW. The same thing is reminiscent of the graph as well.