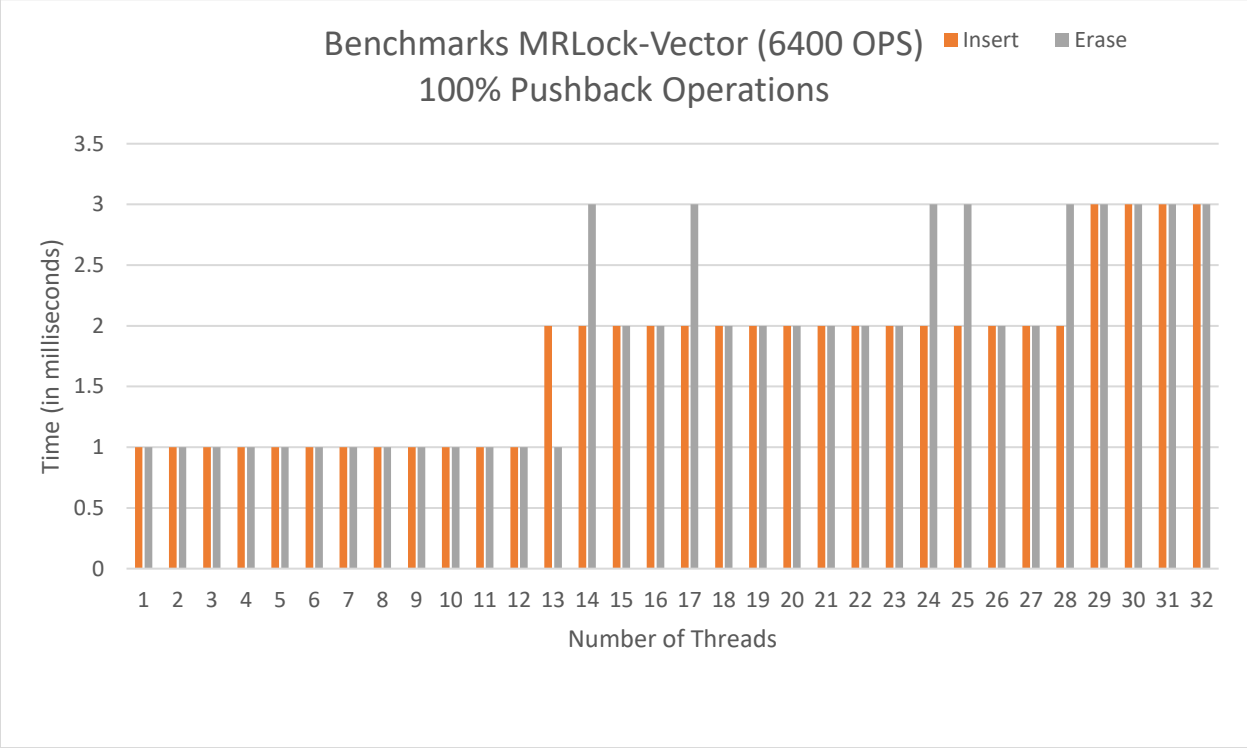
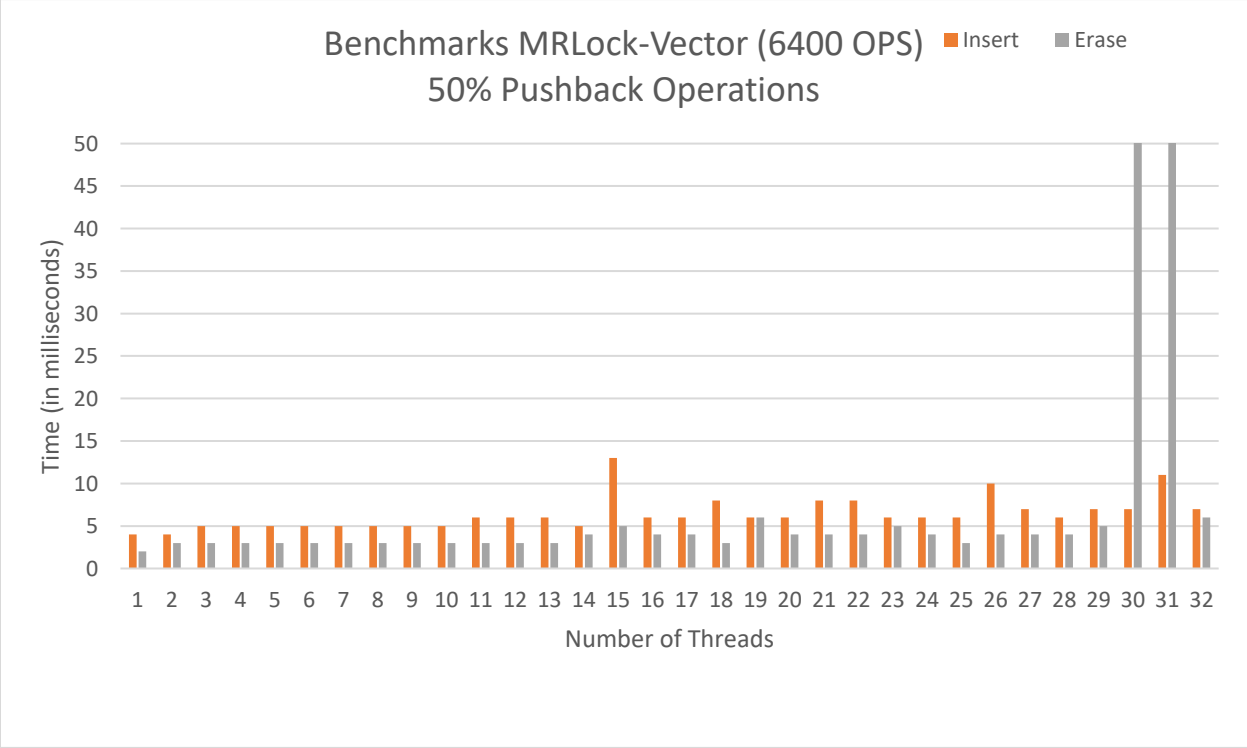
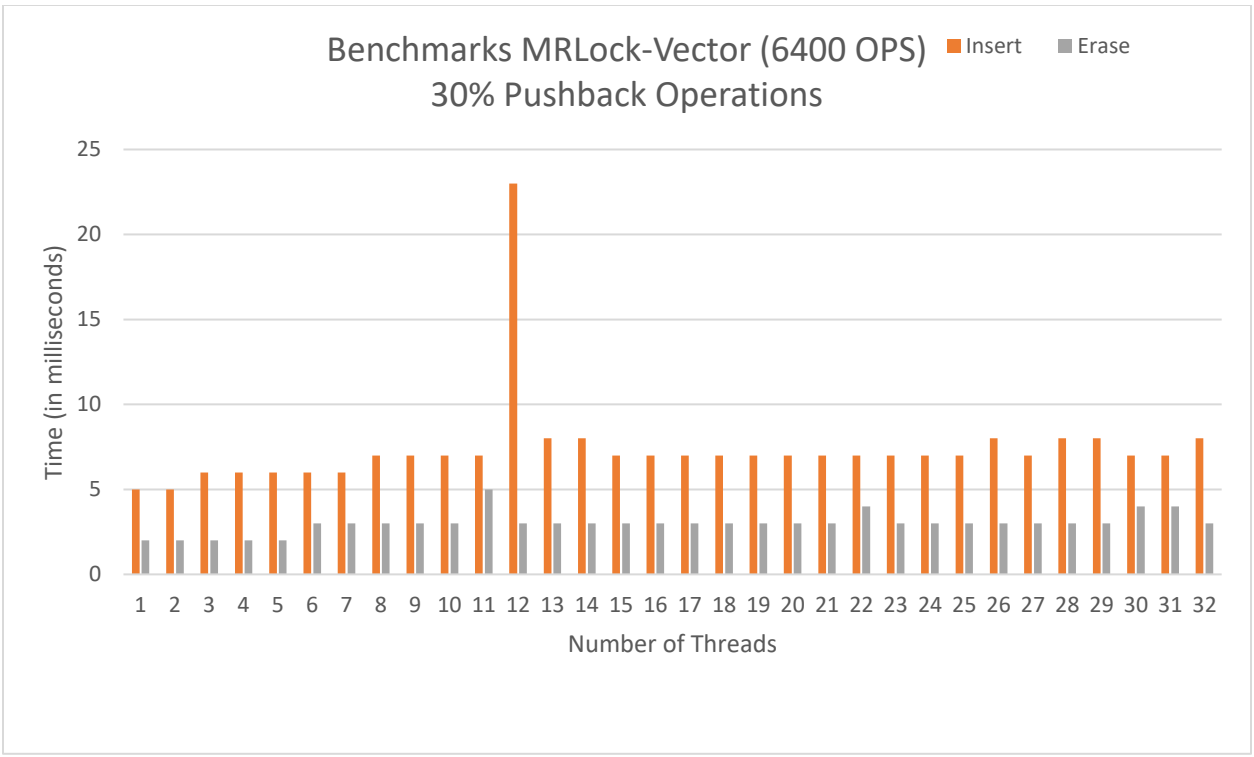
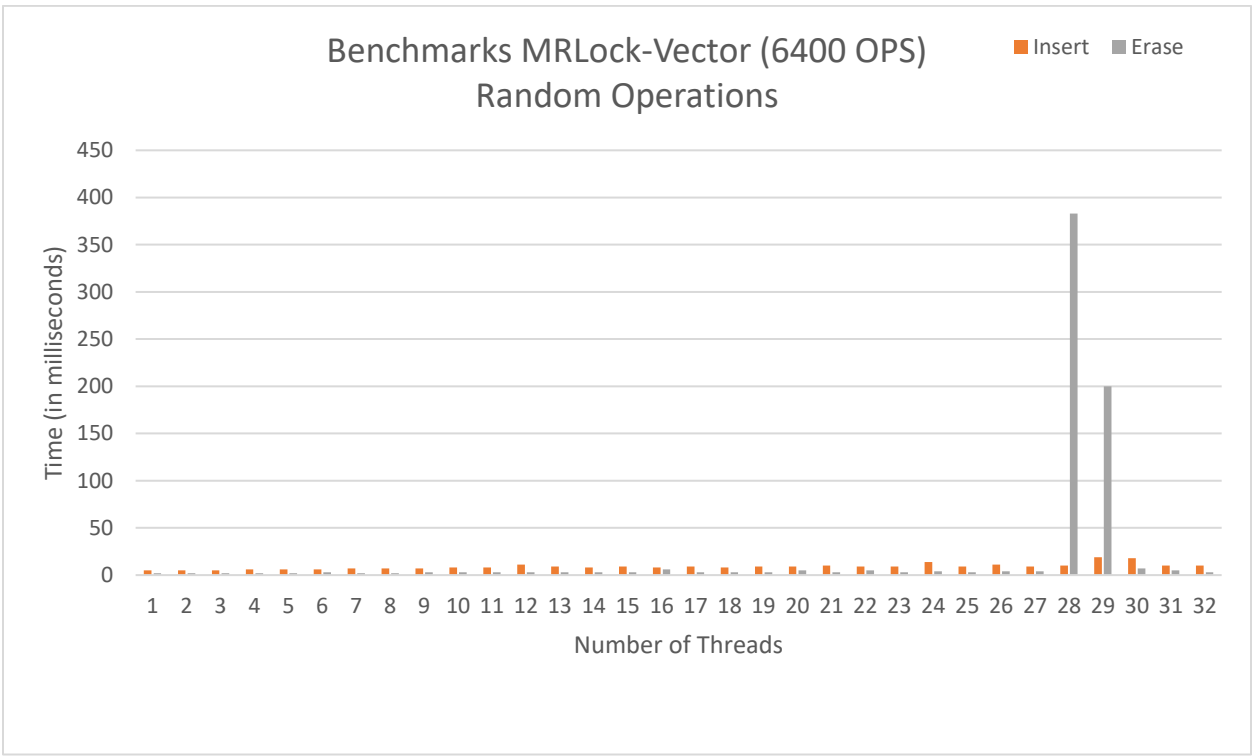
MRLock Vector

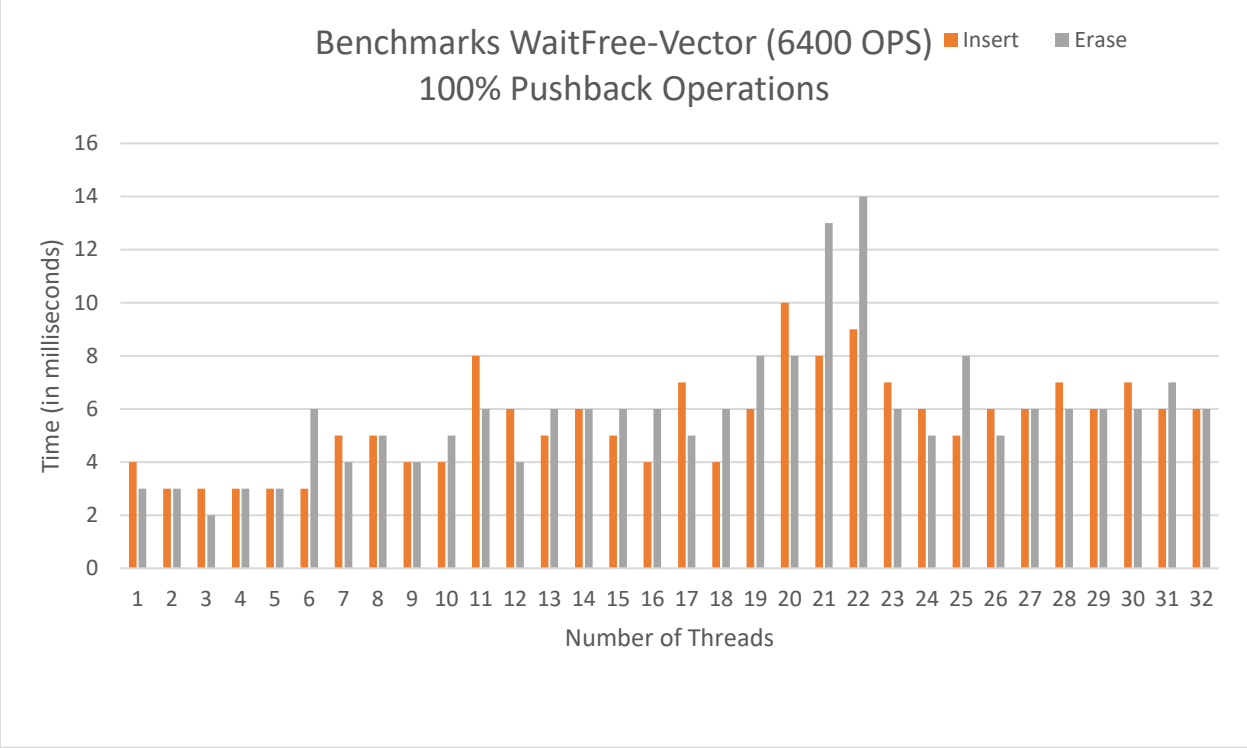


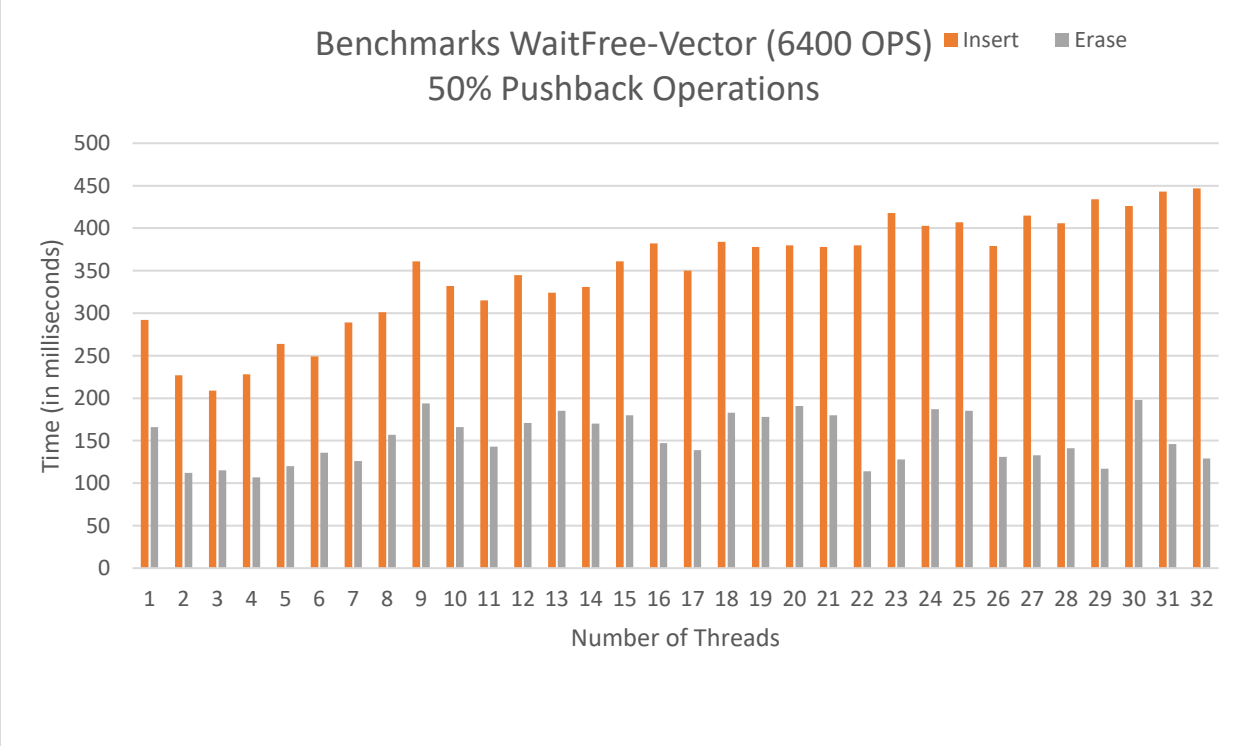


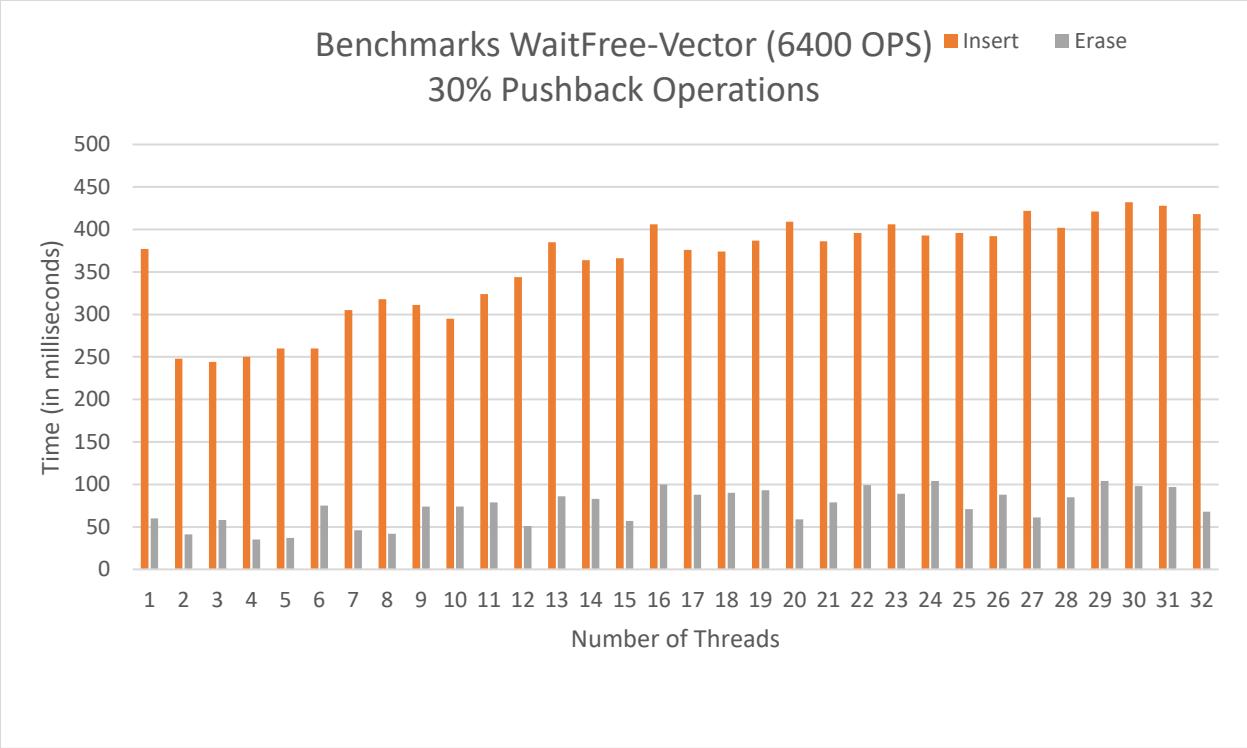

Benchmarks MRLock-Vector (6400 OPS)
30% Pushback Operations
Insert
Erase
Time (in milliseconds)
0
5
10
15
20
25
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Number of Threads

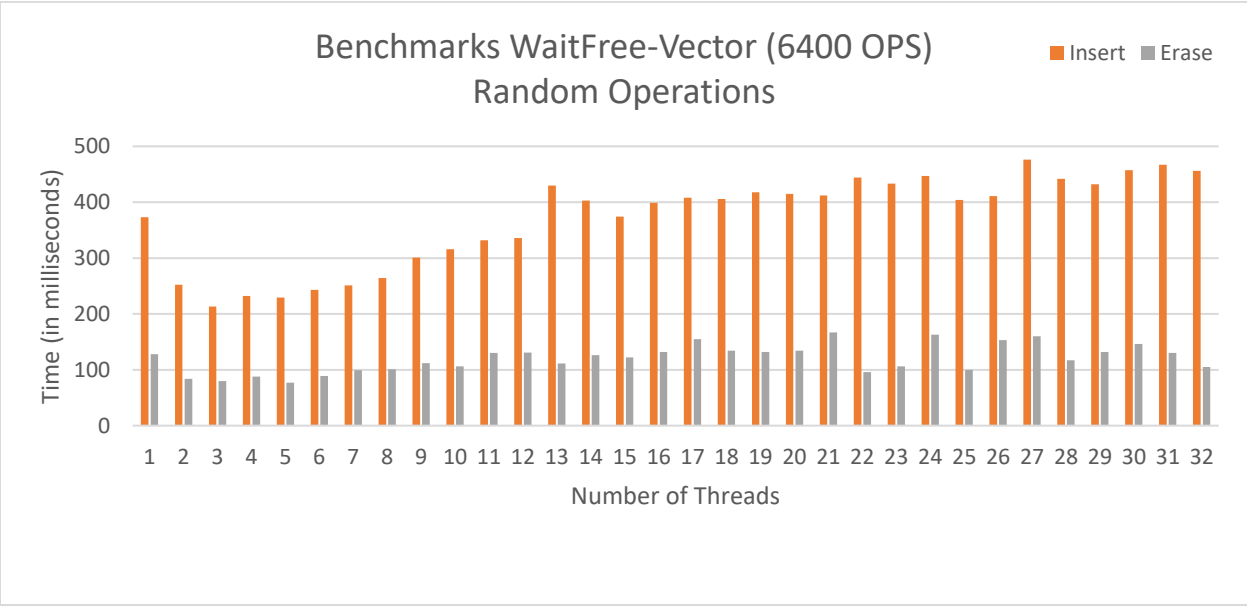

Benchmarks MRLock-Vector (6400 OPS)
Random Operations
Insert
Erase
Time (in milliseconds)
0
50
100
150
200
250
300
350
400
450
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Number of Threads

WaitFree Vector






Benchmarks WaitFree-Vector (6400 OPS)
30% Pushback Operations
Insert
Erase
Time (in milliseconds)
0
50
100
150
200
250
300
350
400
450
500
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Number of Threads


Benchmarks WaitFree-Vector (6400 OPS)
Random Operations
Insert
Erase
Time (in milliseconds)
0
100
200
300
400
500
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Number of Threads

Analysis:

The benchmarks were done by modifying the percentage of push_back() operations for the first three tests for both types of vectors, and then testing a random distribution of all operations. Insert and erase operations were tested separately for both vectors as the wait-free version cannot support both simultaneously.

Under low contention, the lock-based vector performs much better than the wait-free vector; however, as contention increases, the lock-based vector does not scale well. The wait-free vector on the other hand seems to be more consistent and scales appropriately as contention grows.