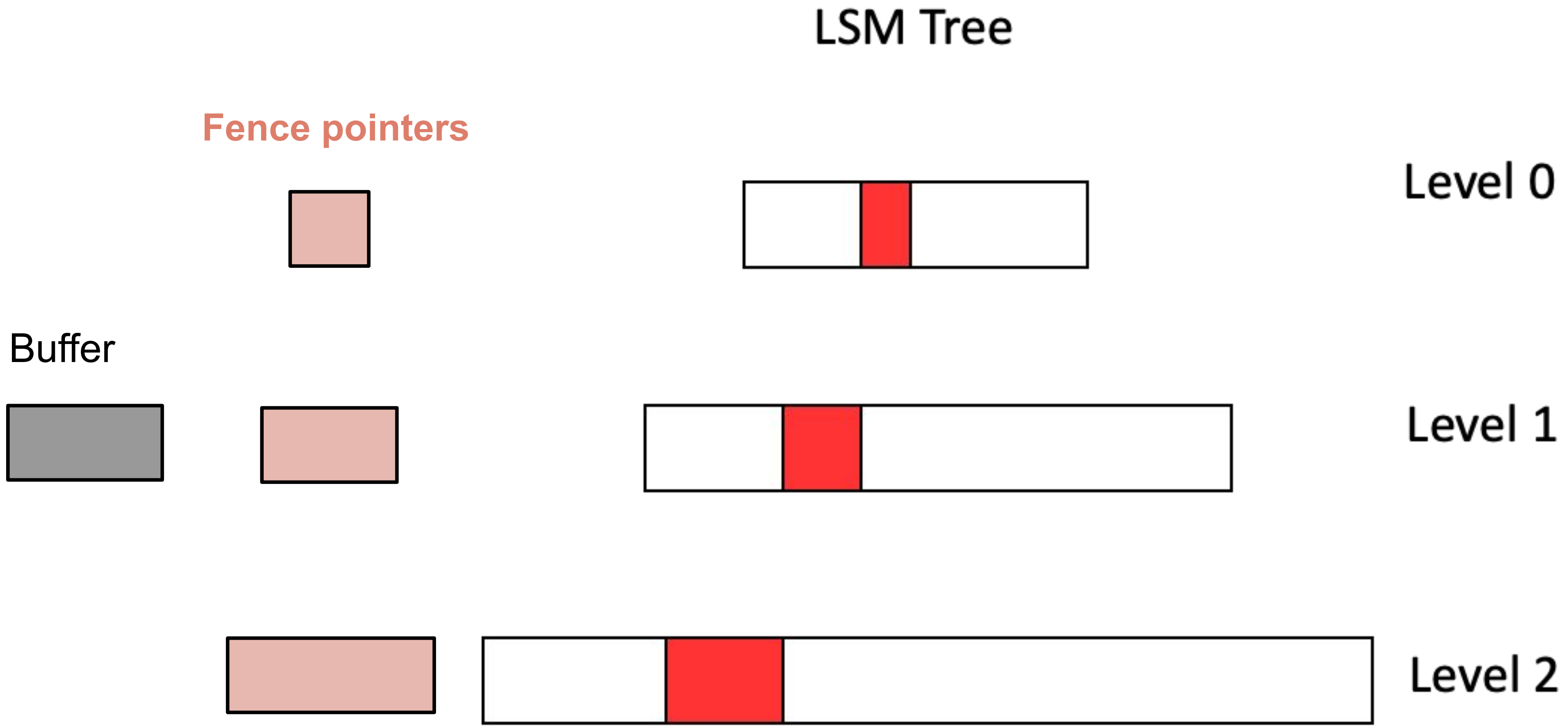
Query-driven compaction in LSM-trees

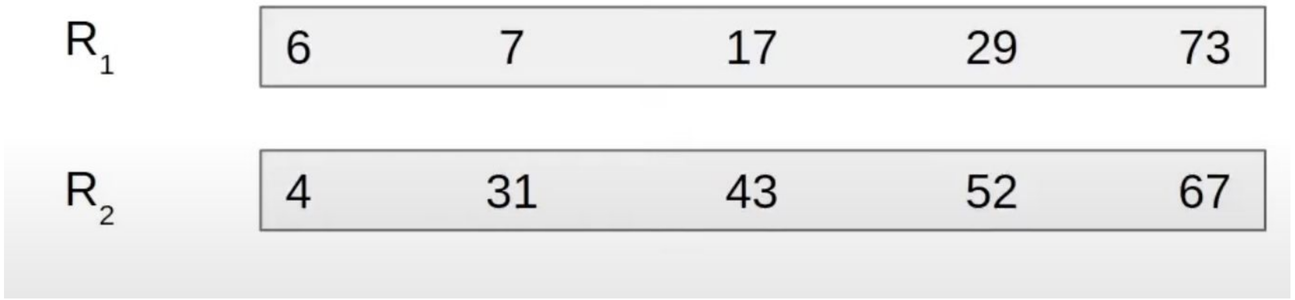
Ye Tian, Peiying Ye, Li Xi

Log-Structured Merge-tree and Range Query



Range Query and its Sorted View

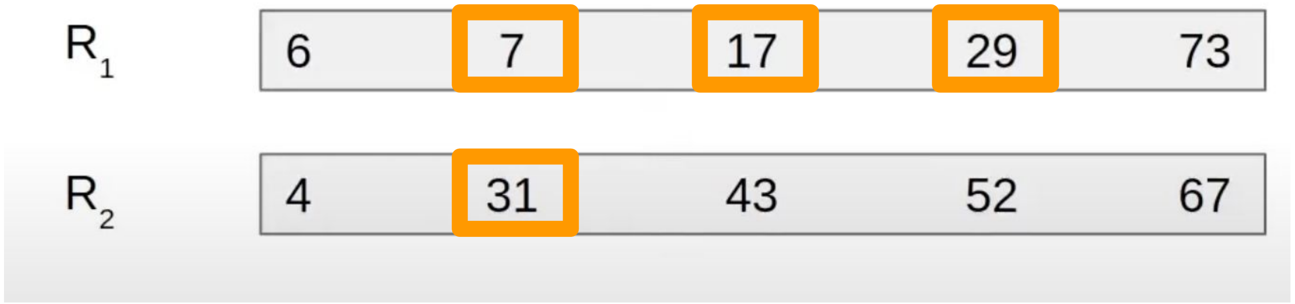
Range query (min = 7, max = 40)



Access each run and find the smallest key >= 7 and <= 40.

Range Query and its Sorted View

Range query (min = 7, max = 40)



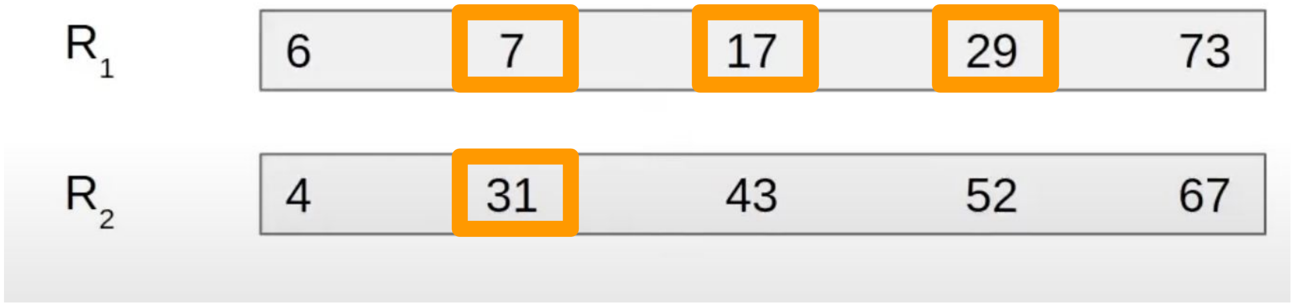
Access each run and find the smallest key >= 7 and <= 40.

→

Compare, build heap, and output the smallest.

Range Query and its Sorted View

Range query (min = 7, max = 40)



Access each run and find the smallest key >= 7 and <= 40.

Compare, build heap, and output the smallest.

Runtime Sort-merge completed.

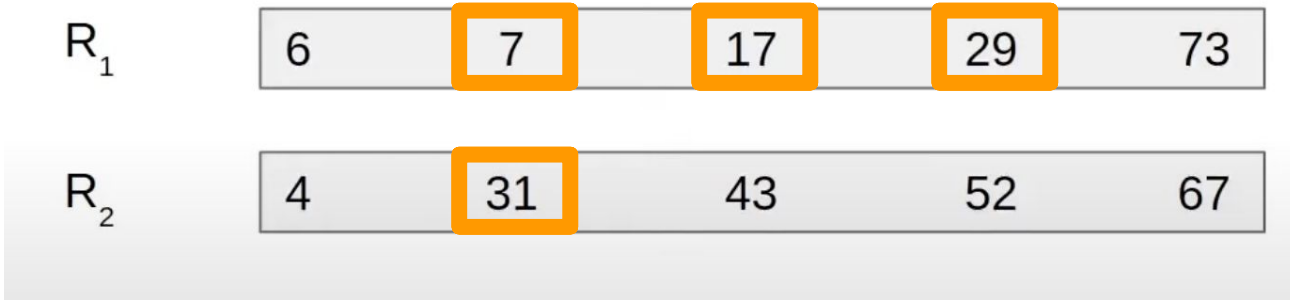
7, 17, 29, 31

Discarded after range query completed.

Duplicate Sort-merge next time!

Research Motivation

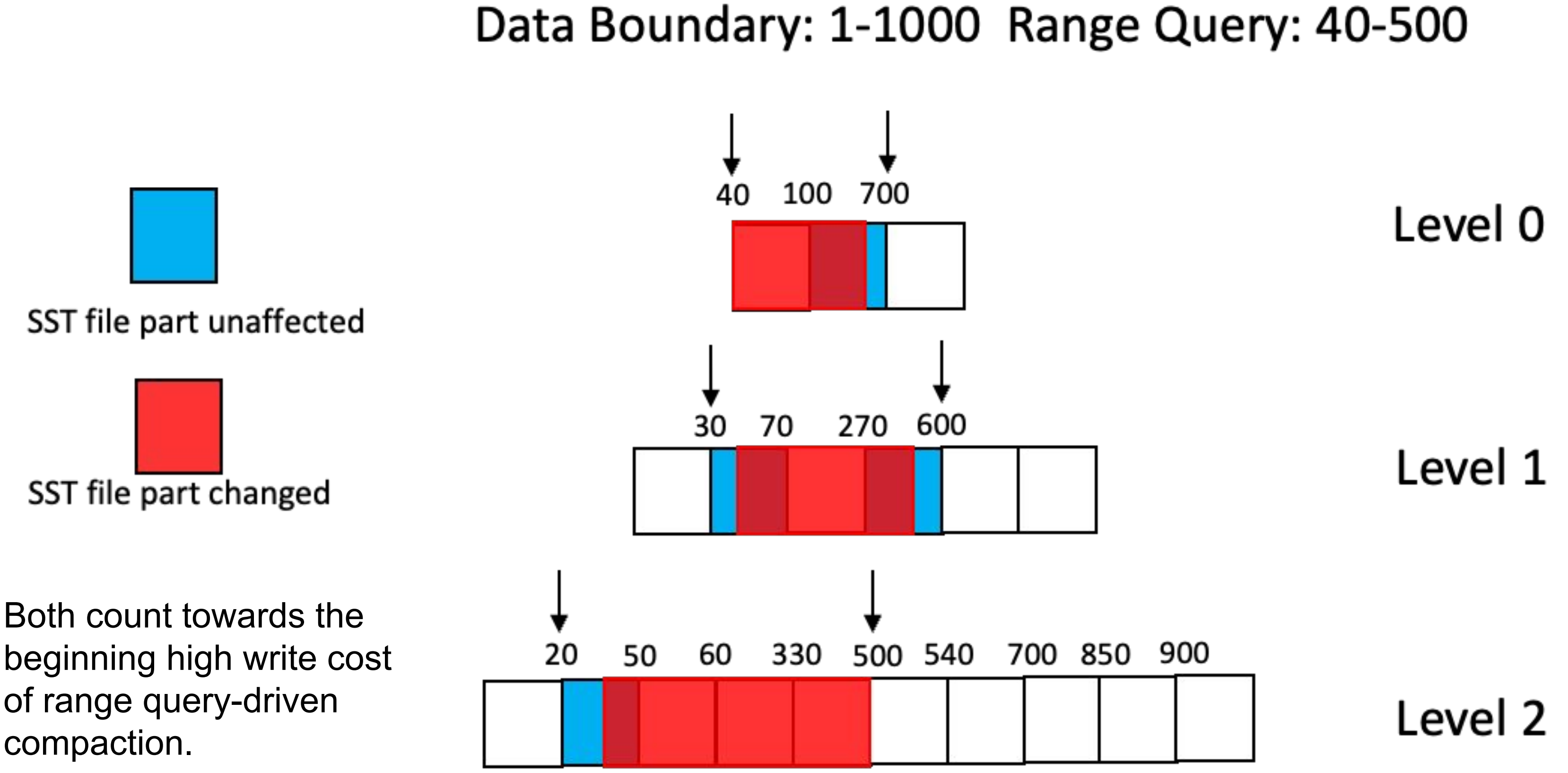
Range query (min = 7, max = 40)



R_1	6	7	17	29	73
R_2	4	31	43	52	67

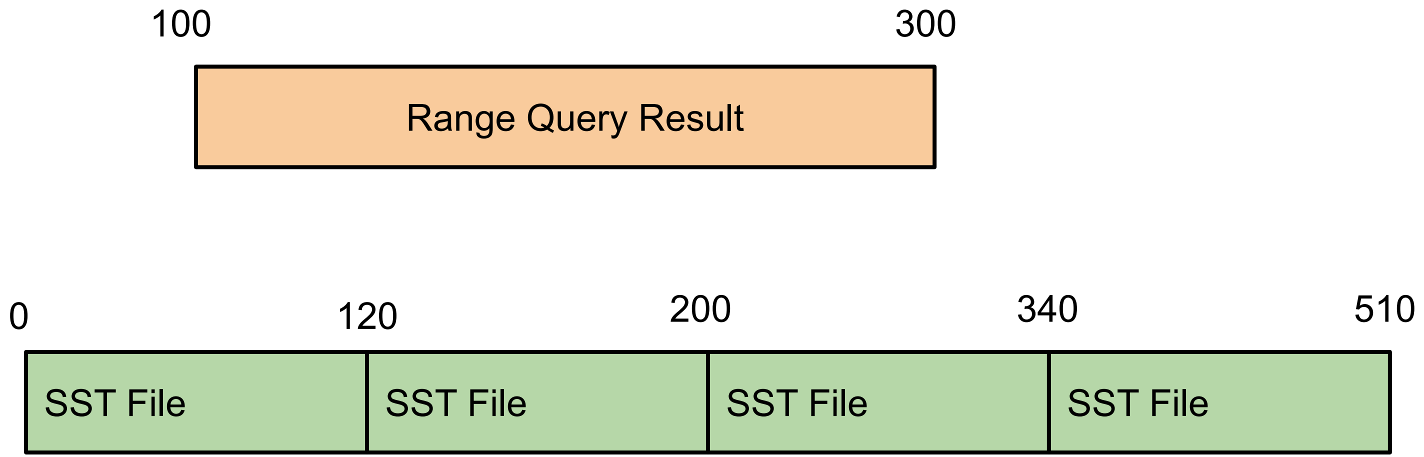
7, 17, 29, 31 → **Duplicate Sort-merge next time!** → **How can we optimize it?**

How does range-query-driven compaction work?

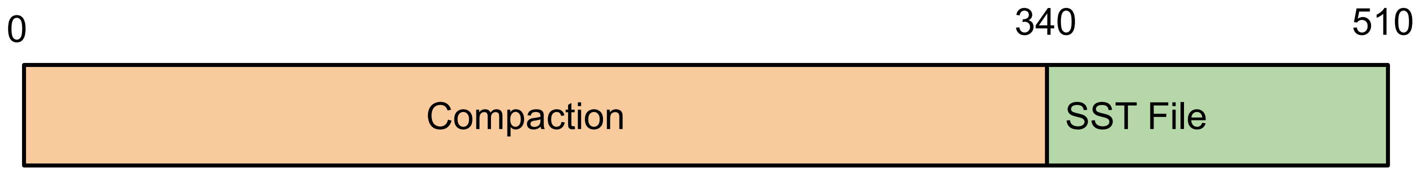


Both count towards the beginning high write cost of range query-driven compaction.

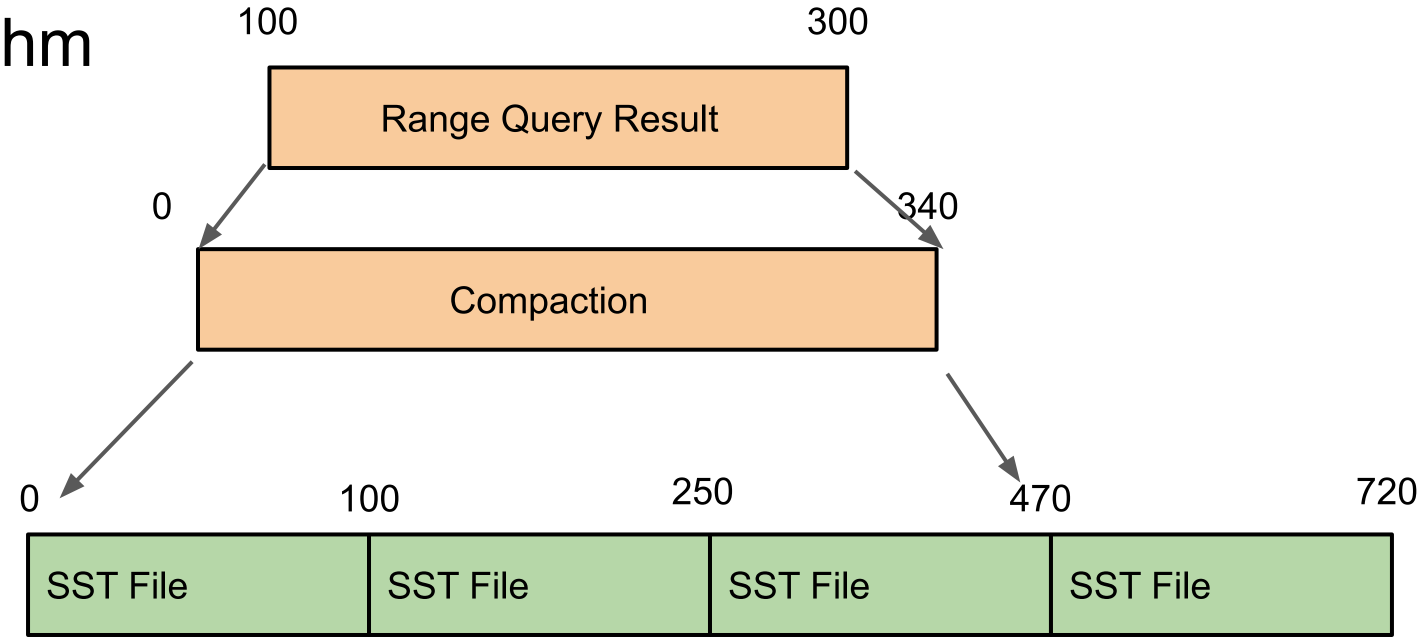
Algorithm


100
300
Range Query Result
0
120
200
340
510
SST File
SST File
SST File
SST File

Algorithm

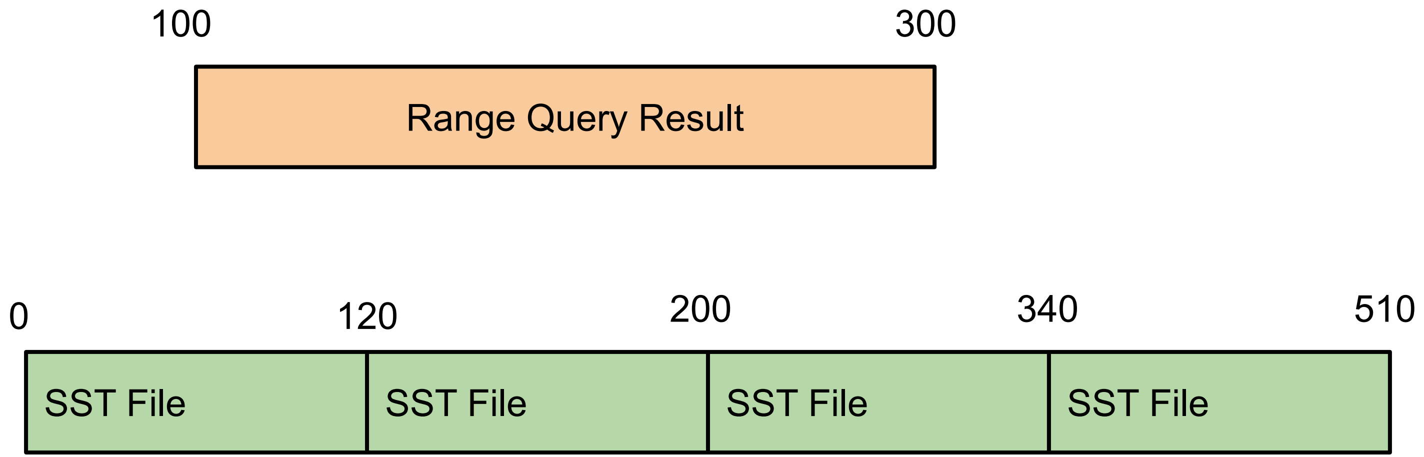


Algorithm

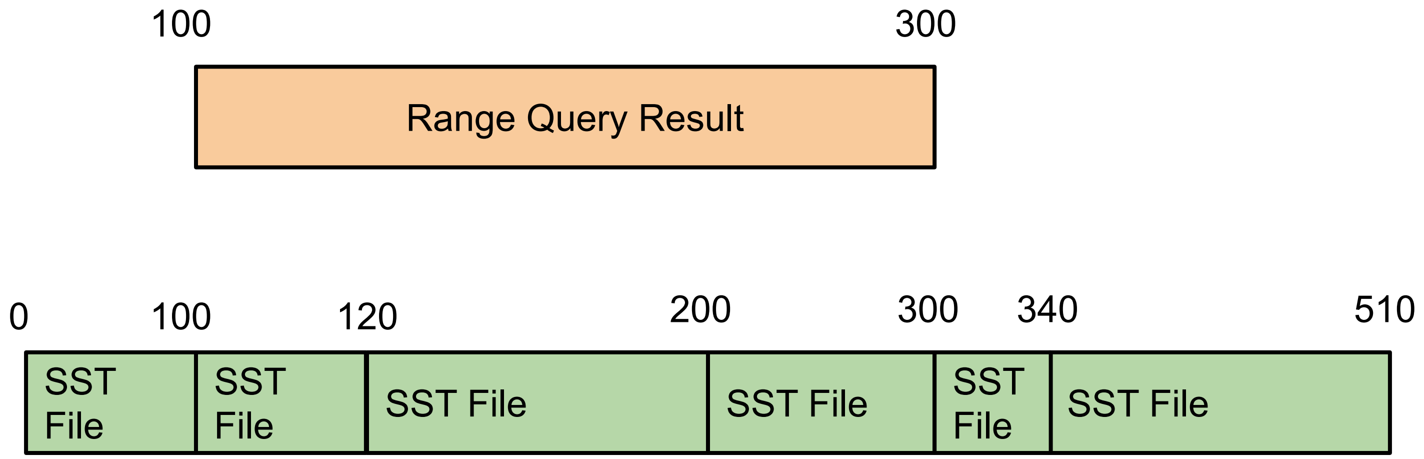


Don’t work for multi-layer compaction
Compaction range keeps growing

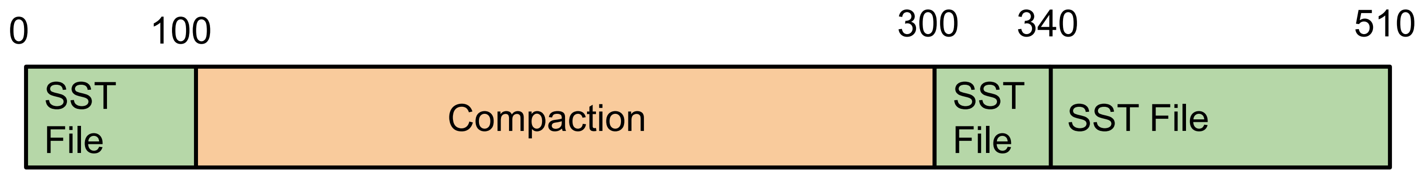
Algorithm


100
300
Range Query Result
0
120
200
340
510
SST File
SST File
SST File
SST File

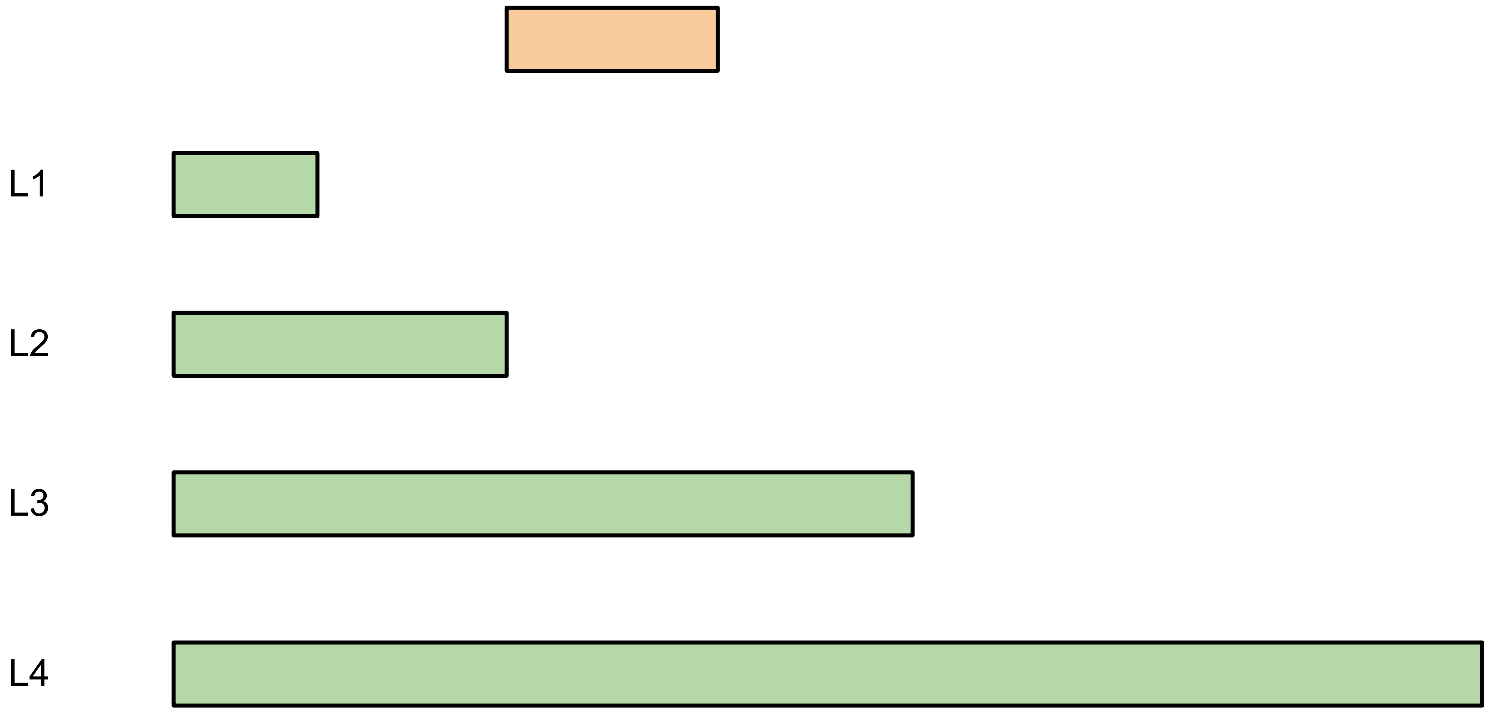
Algorithm


100
300
Range Query Result
0
100
120
200
300
340
510
SST File
SST File
SST File
SST File
SST File
SST File

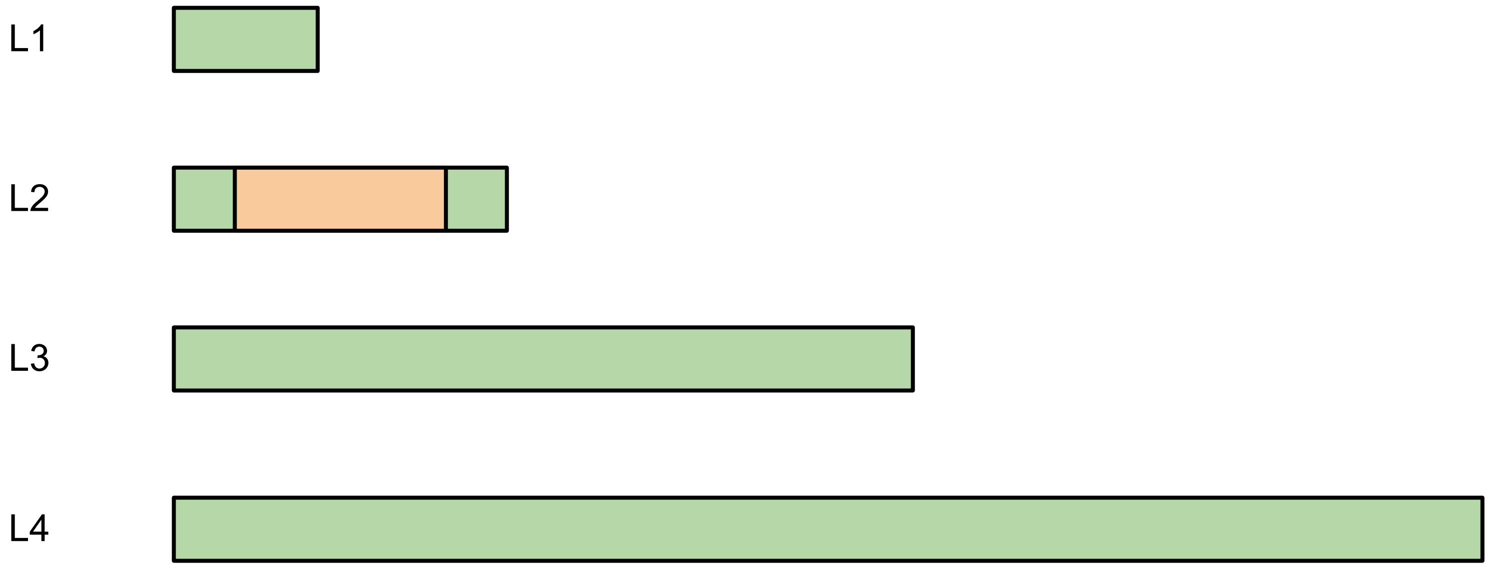
Algorithm



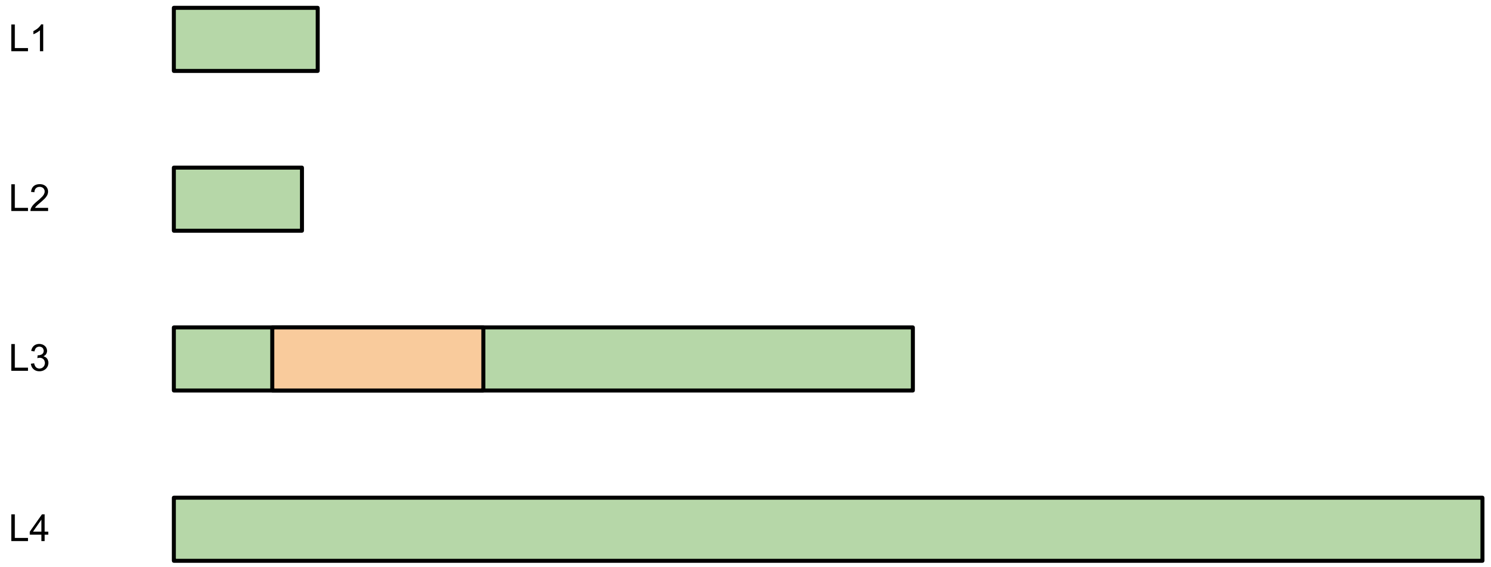
Algorithm



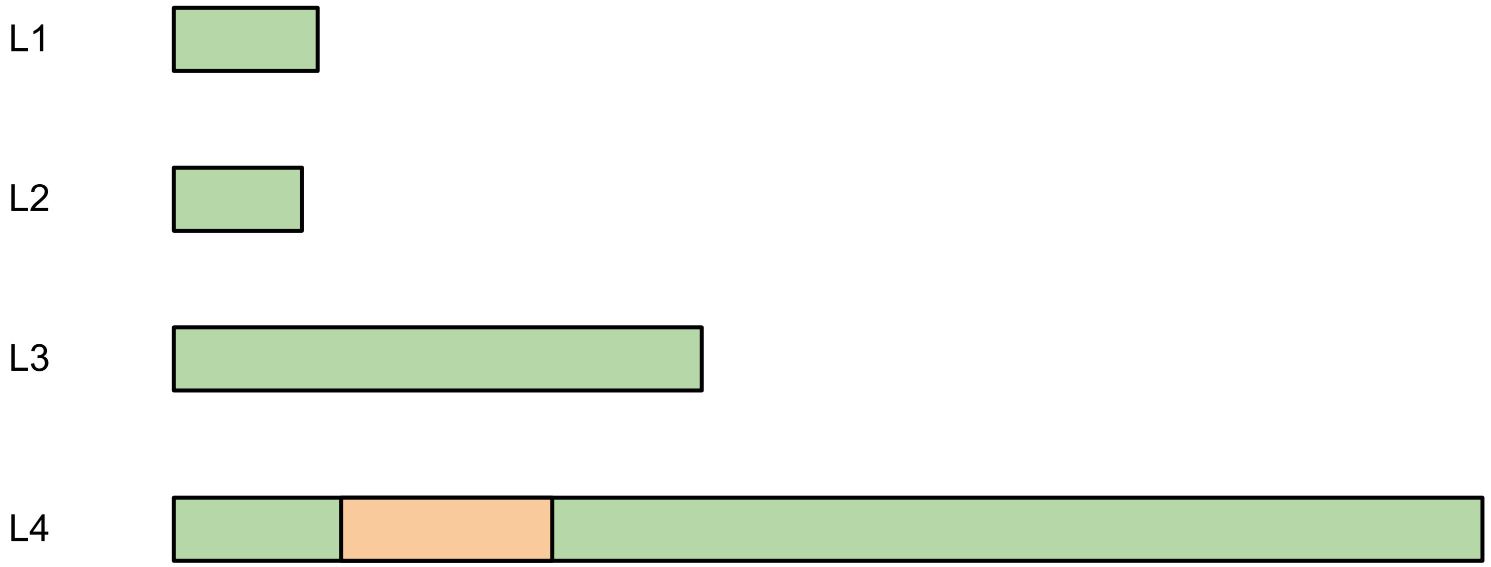
Algorithm



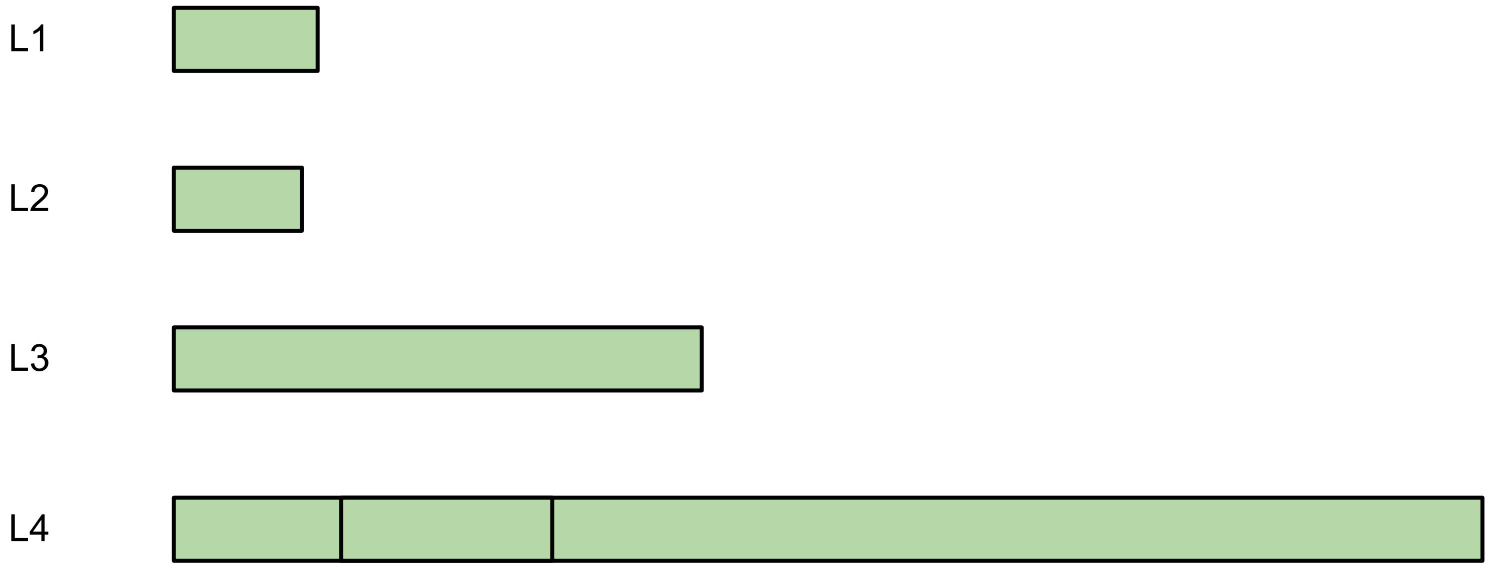
Algorithm



Algorithm

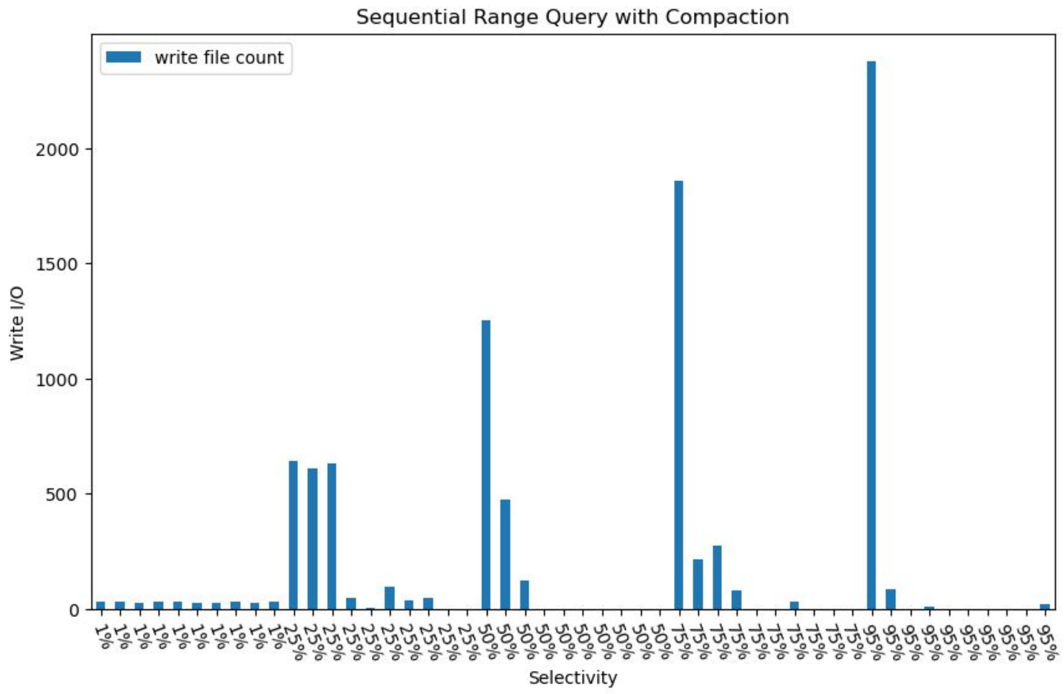


Algorithm

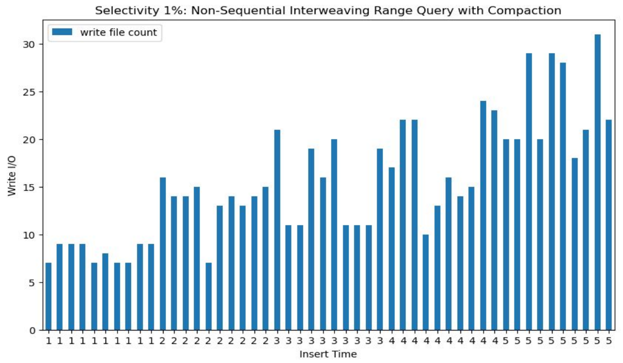


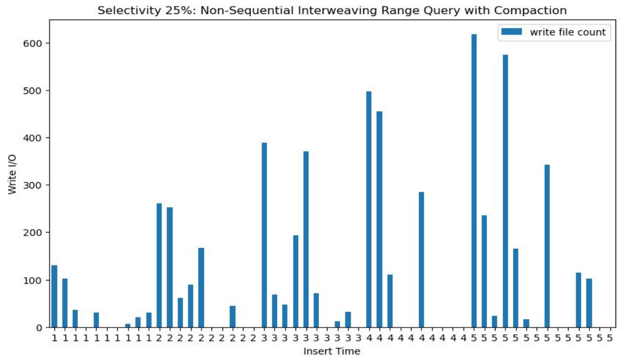
Benchmark Experiments (100k Inserts)

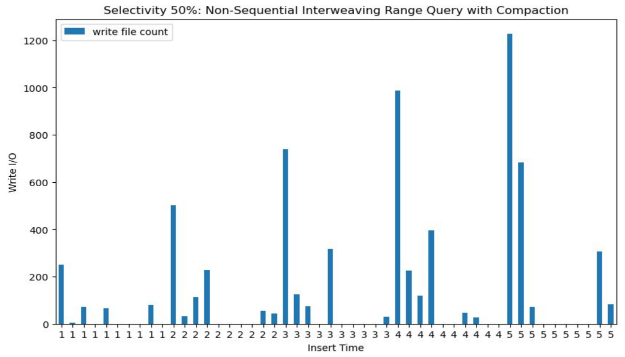
Sequential Workload (50RQ)

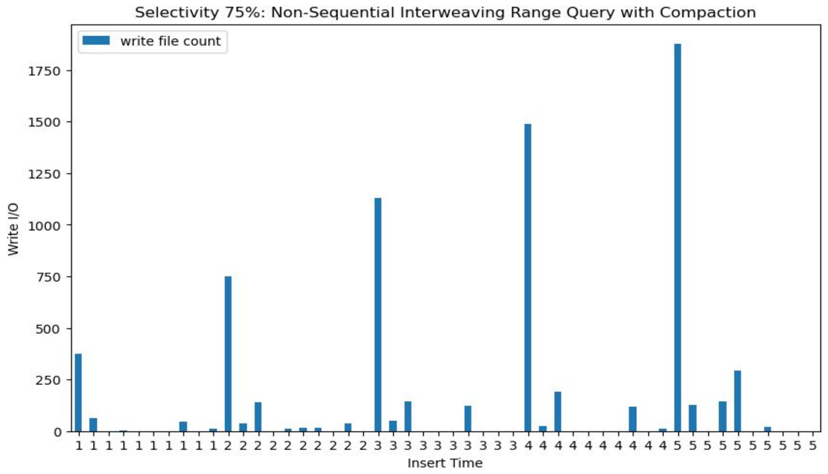


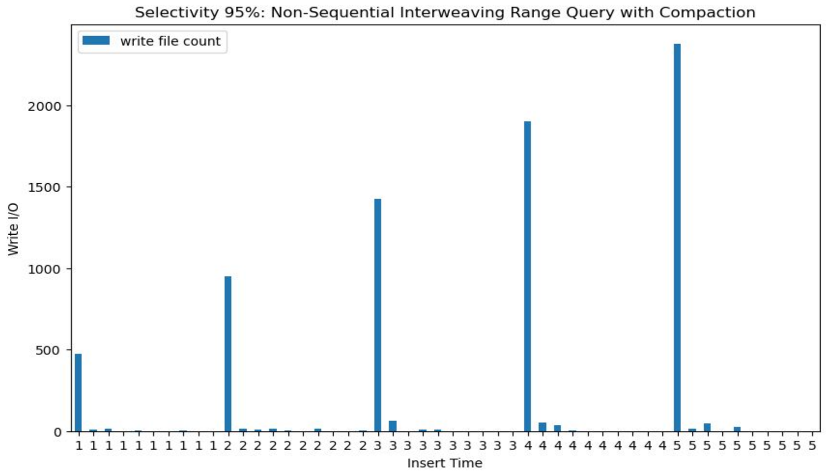
Non-sequential Workload (5 times inserts, 100RQ each time)



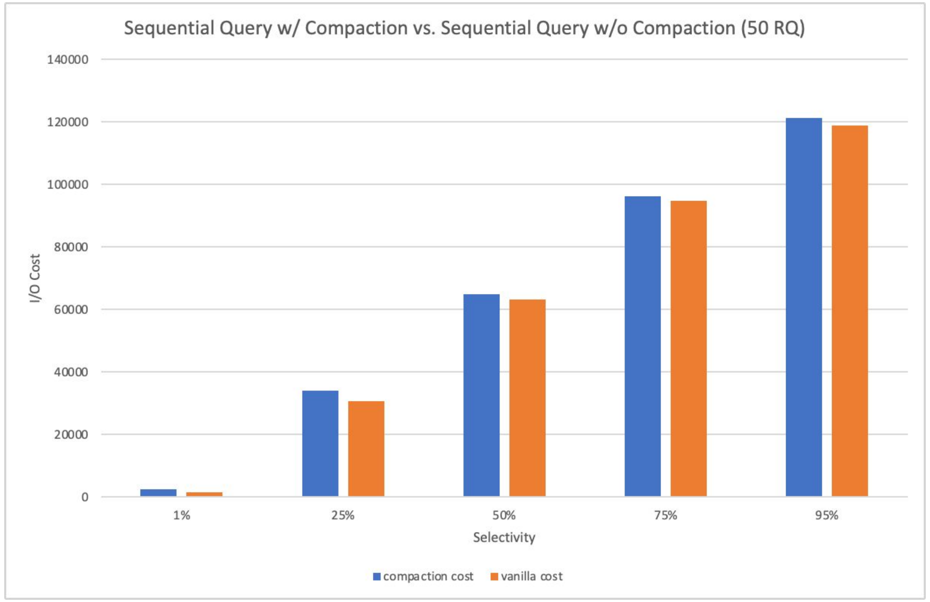


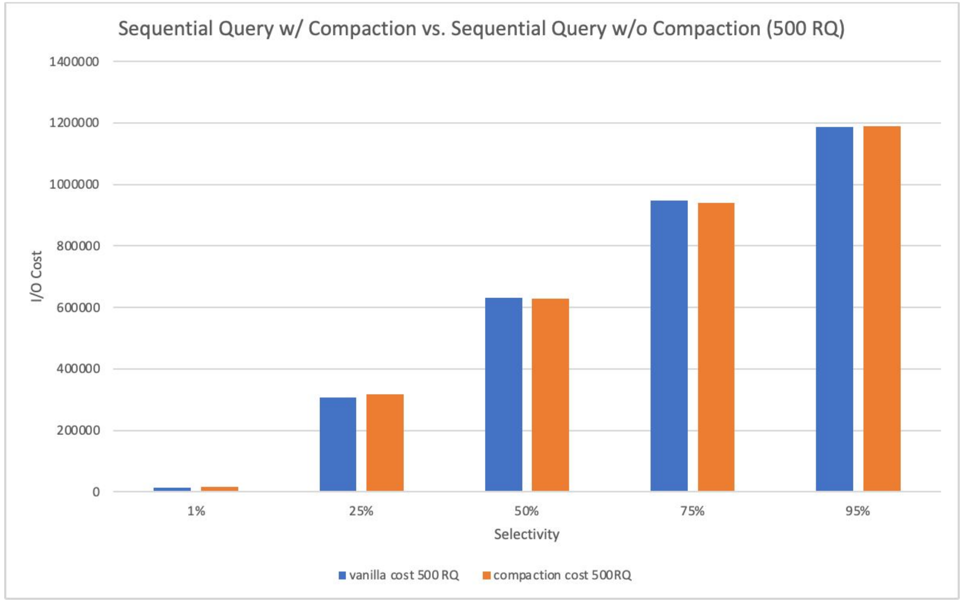






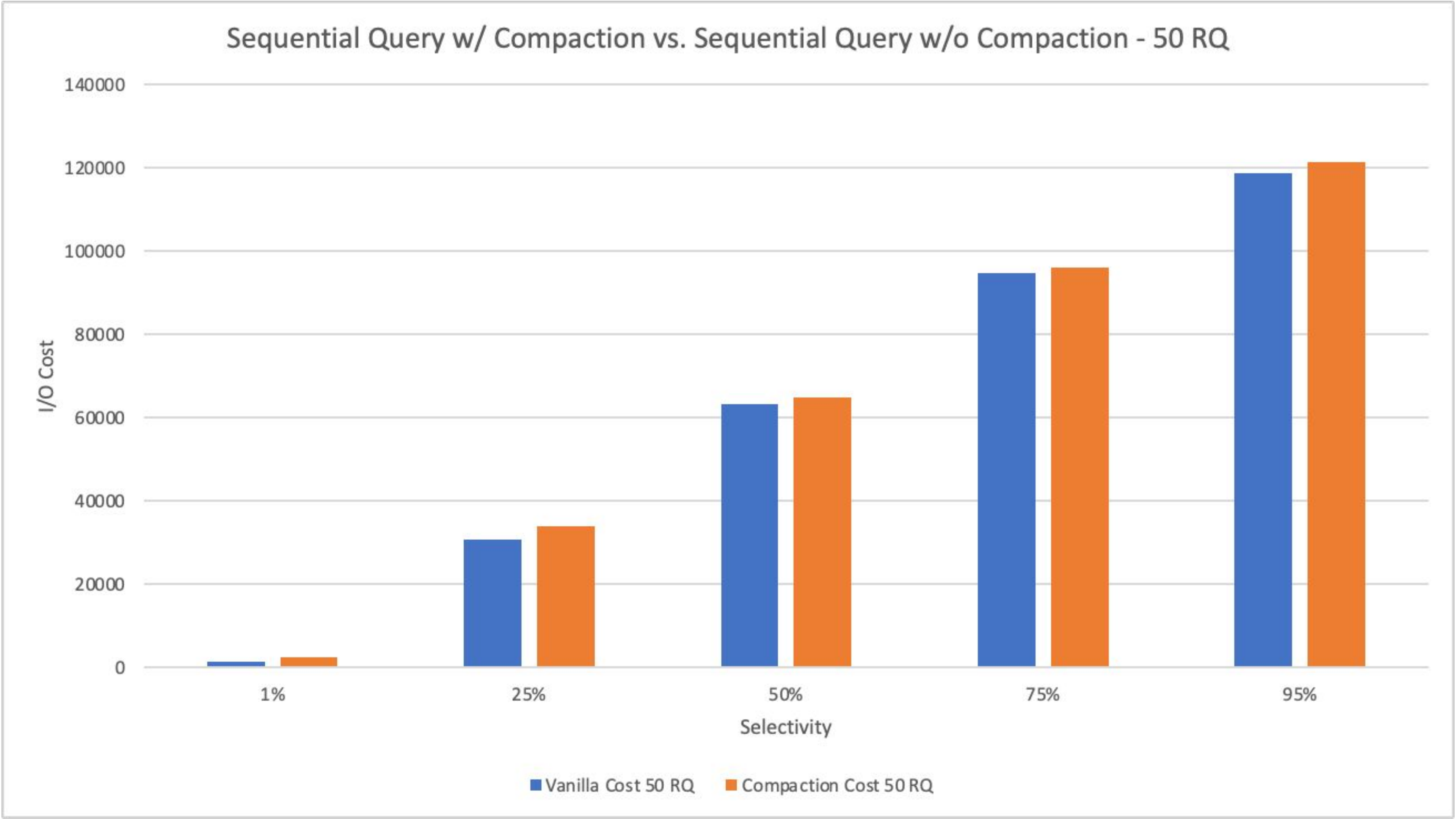
Comparison

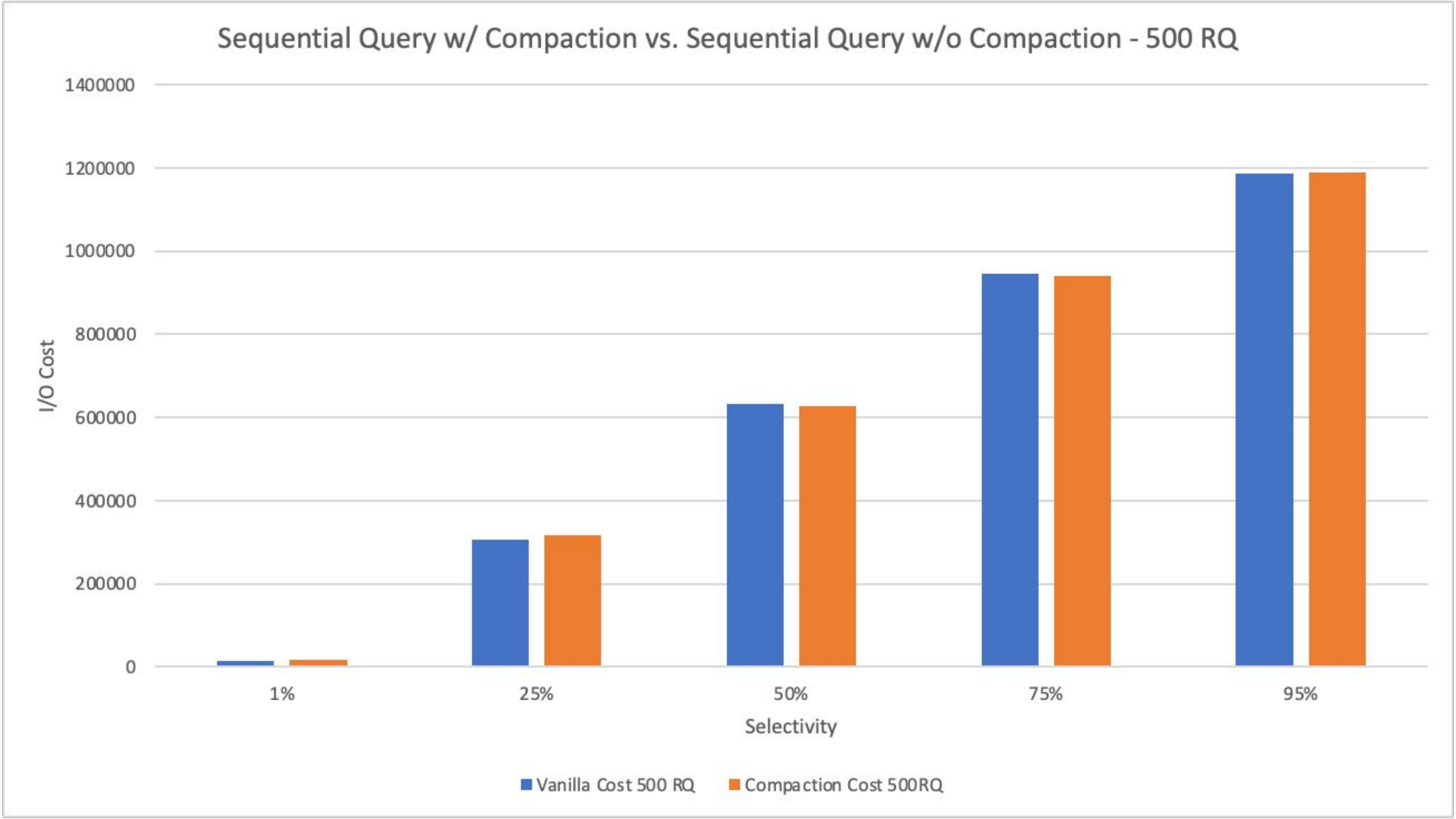

Sequential Query w/ Compaction vs. Sequential Query w/o Compaction (50 RQ)
140000
120000
100000
80000
60000
40000
20000
0
I/O Cost
1%
25%
50%
75%
95%
Selectivity
■ compaction cost
■ vanilla cost


Sequential Query w/ Compaction vs. Sequential Query w/o Compaction (500 RQ)
1400000
1200000
1000000
800000
600000
400000
200000
0
I/O Cost
1%
25%
50%
75%
95%
Selectivity
■ vanilla cost 500 RQ
■ compaction cost 500RQ

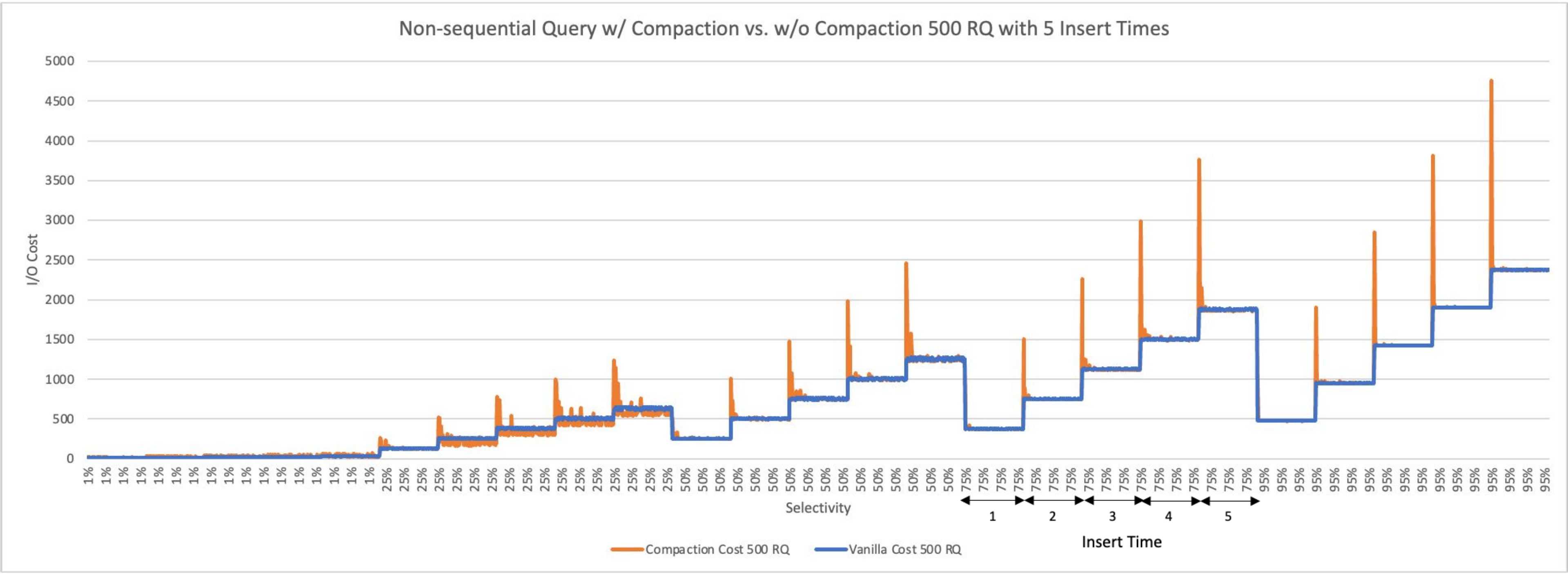
Challenges and Future Evaluations

- Further experiments on comparing non-sequential workload I/O for vanilla implementation and query-driven-compaction implementation
- Further experiments on ‘QueryDrivenCompactionSelectivity’ to figure out the trend and see if we can find a sweet spot

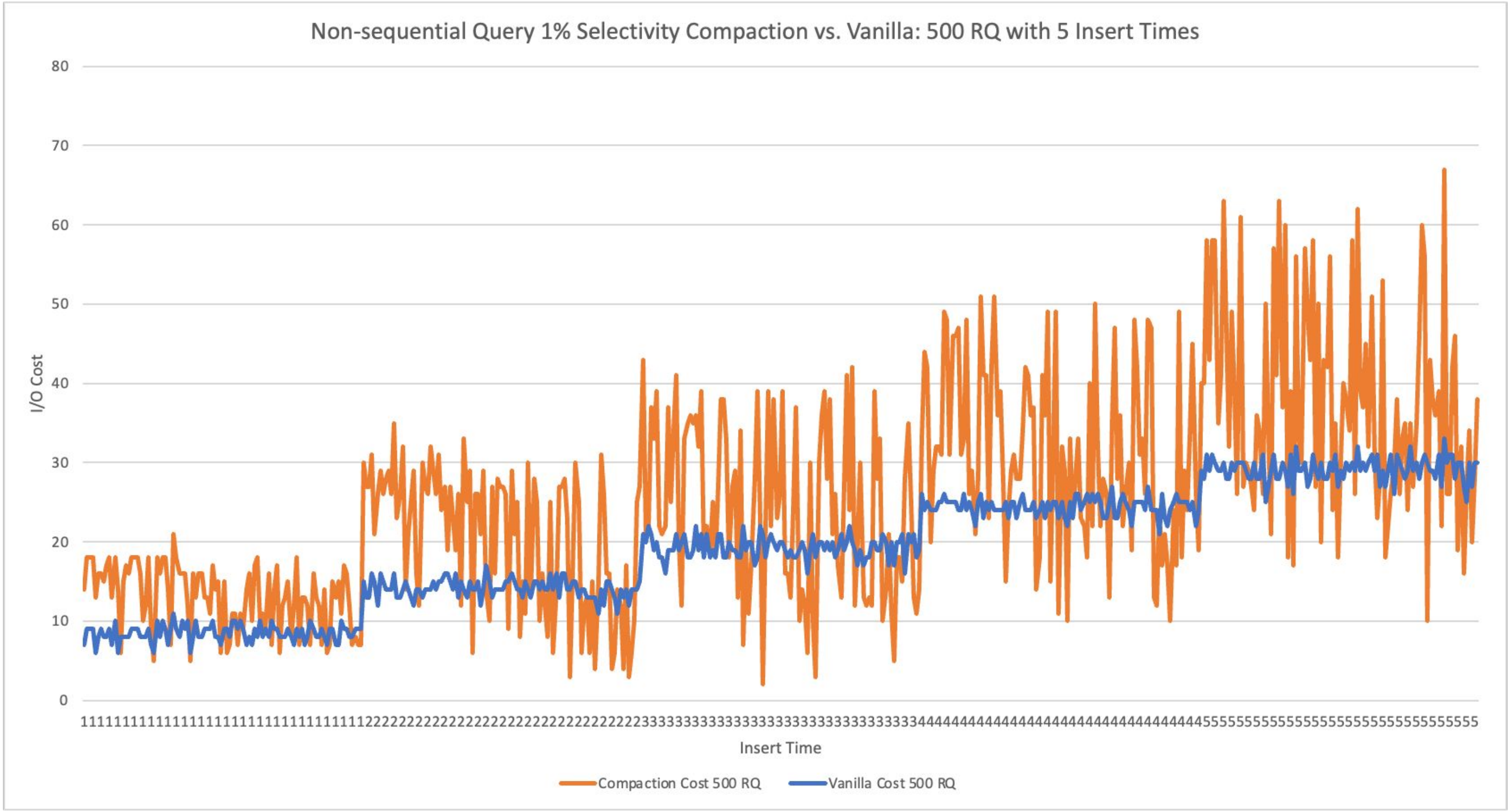

Sequential Query w/ Compaction vs. Sequential Query w/o Compaction - 50 RQ
140000
120000
100000
80000
60000
40000
20000
0
I/O Cost
1%
25%
50%
75%
95%
Selectivity
Vanilla Cost 50 RQ
Compaction Cost 50 RQ


Sequential Query w/ Compaction vs. Sequential Query w/o Compaction - 500 RQ
1400000
1200000
1000000
800000
600000
400000
200000
0
I/O Cost
1%
25%
50%
75%
95%
Selectivity
Vanilla Cost 500 RQ
Compaction Cost 500RQ

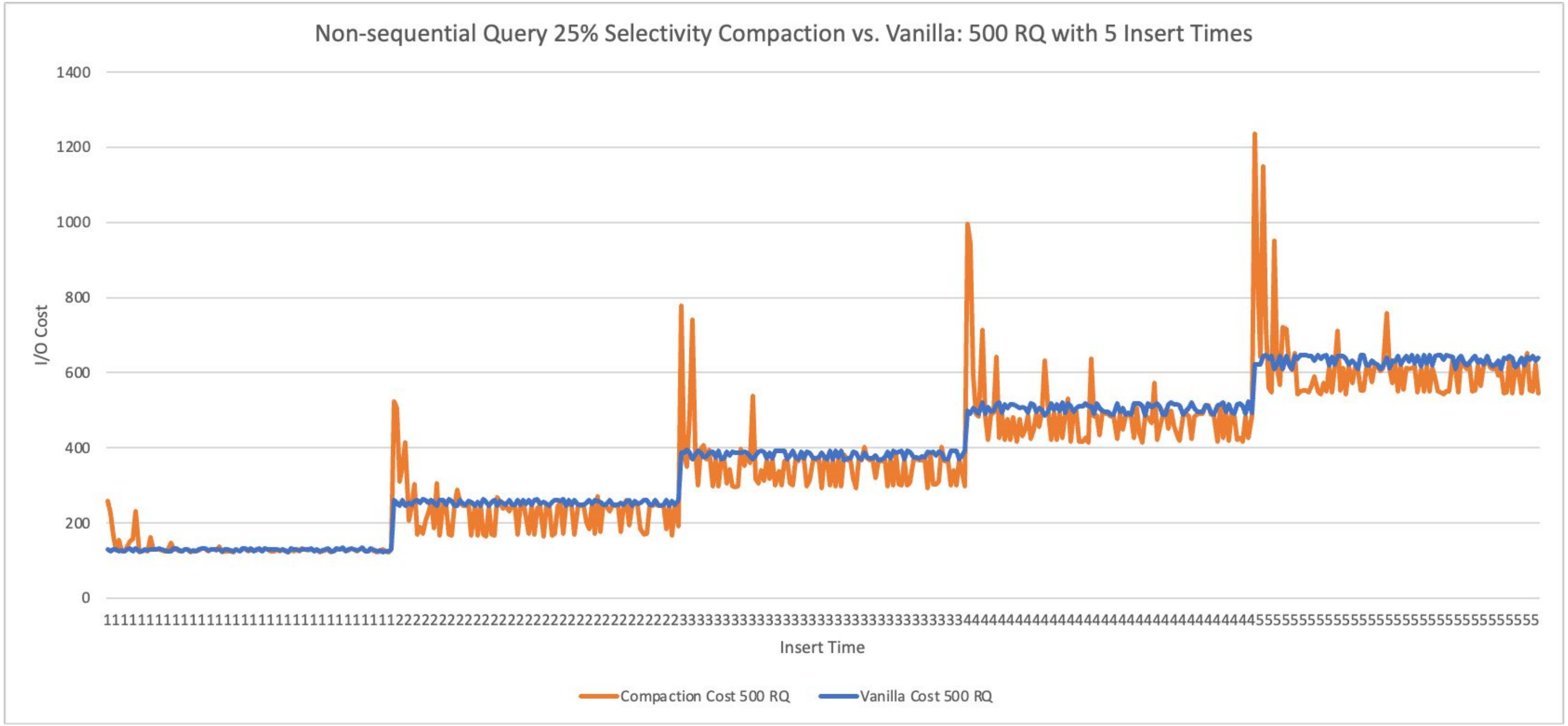
Non-sequential Query: 500 RQ, 5 times insert



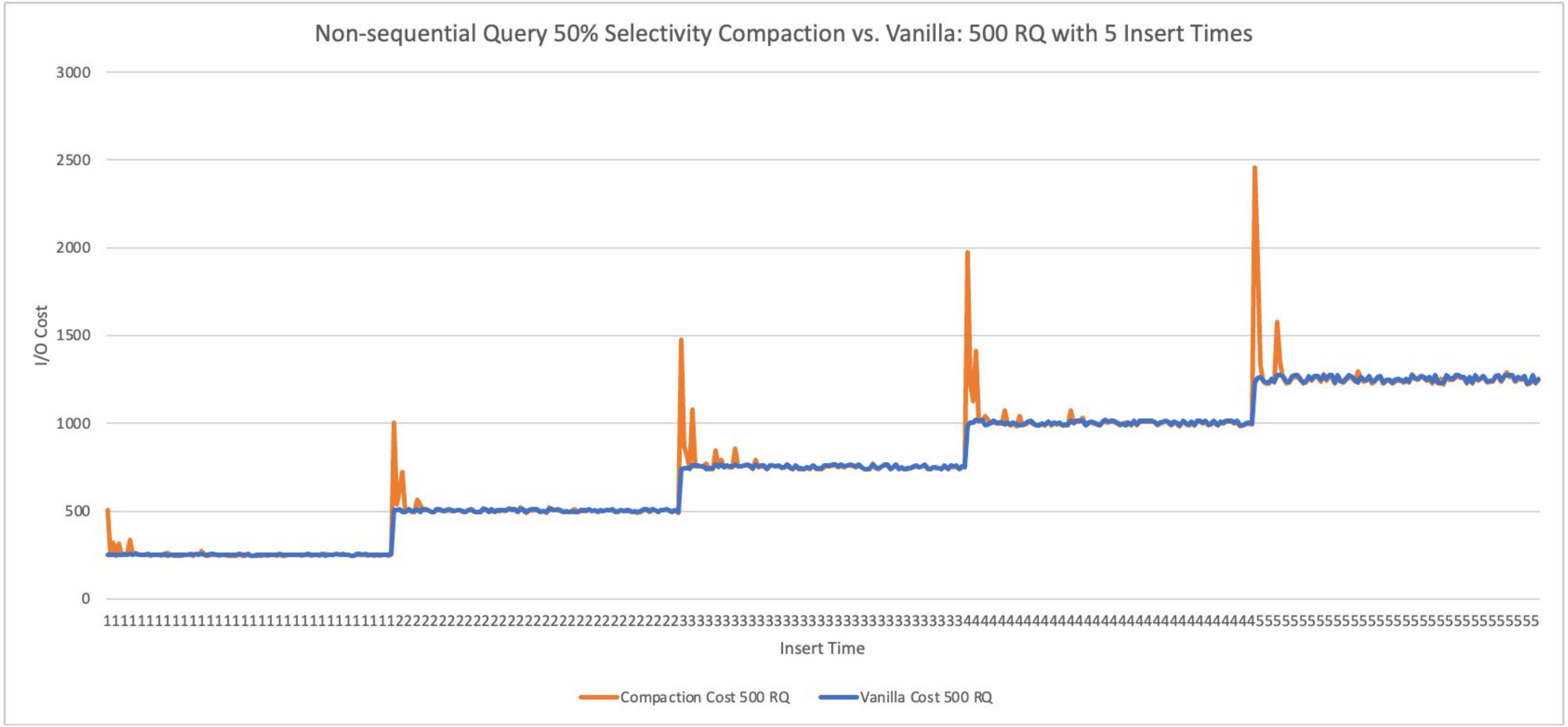
Non-sequential Query: 500 RQ, 5 times insert -1% Selectivity



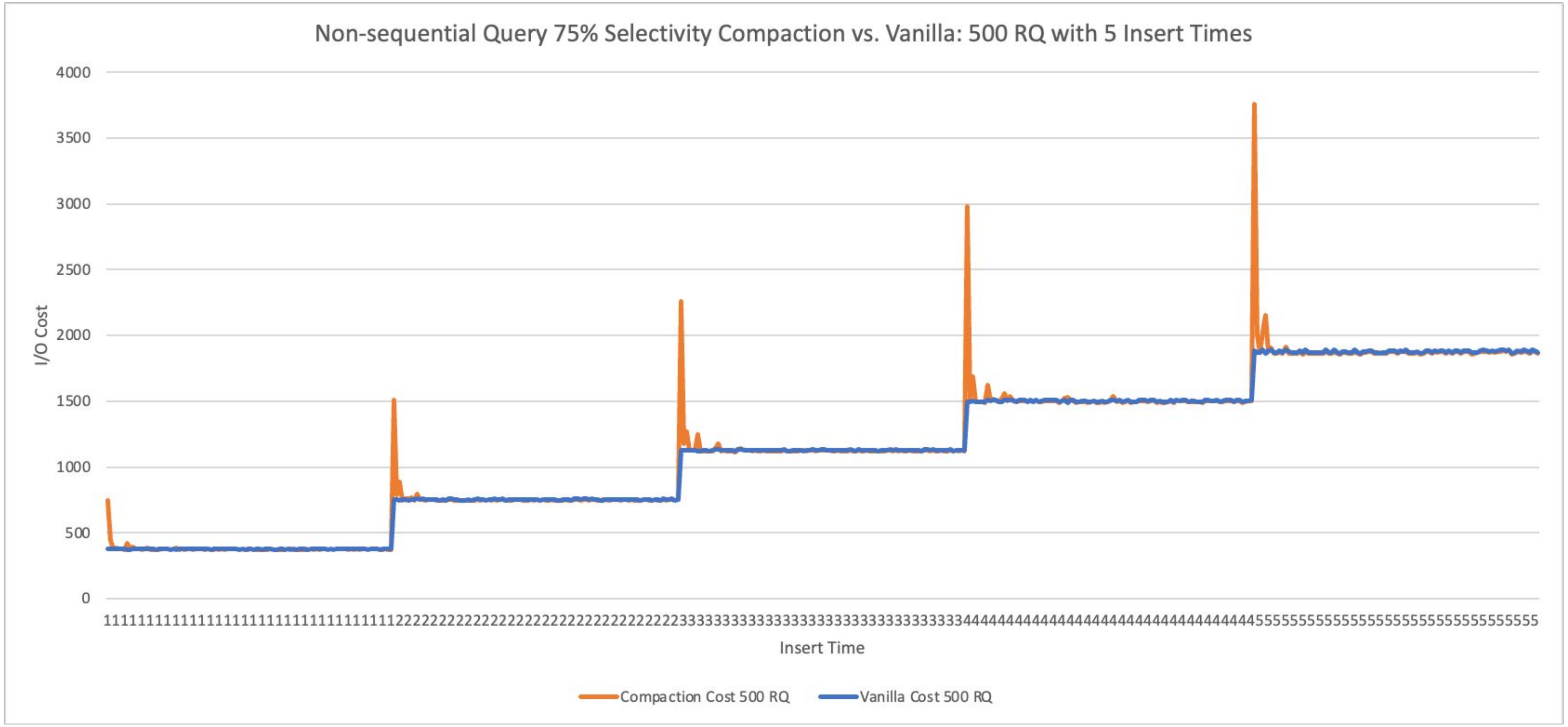
Non-sequential Query: 500 RQ, 5 times insert -25% Selectivity



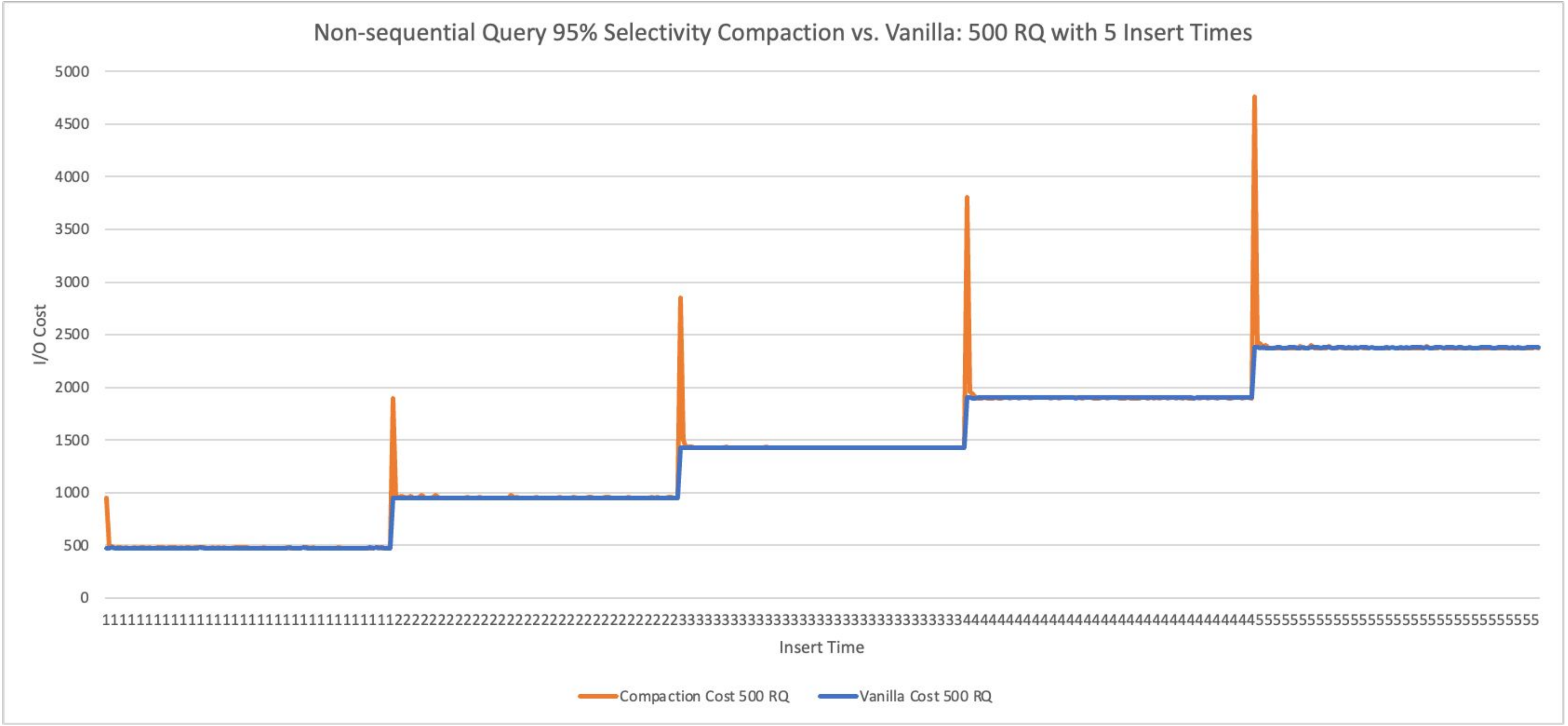
Non-sequential Query: 500 RQ, 5 times insert -50% Selectivity



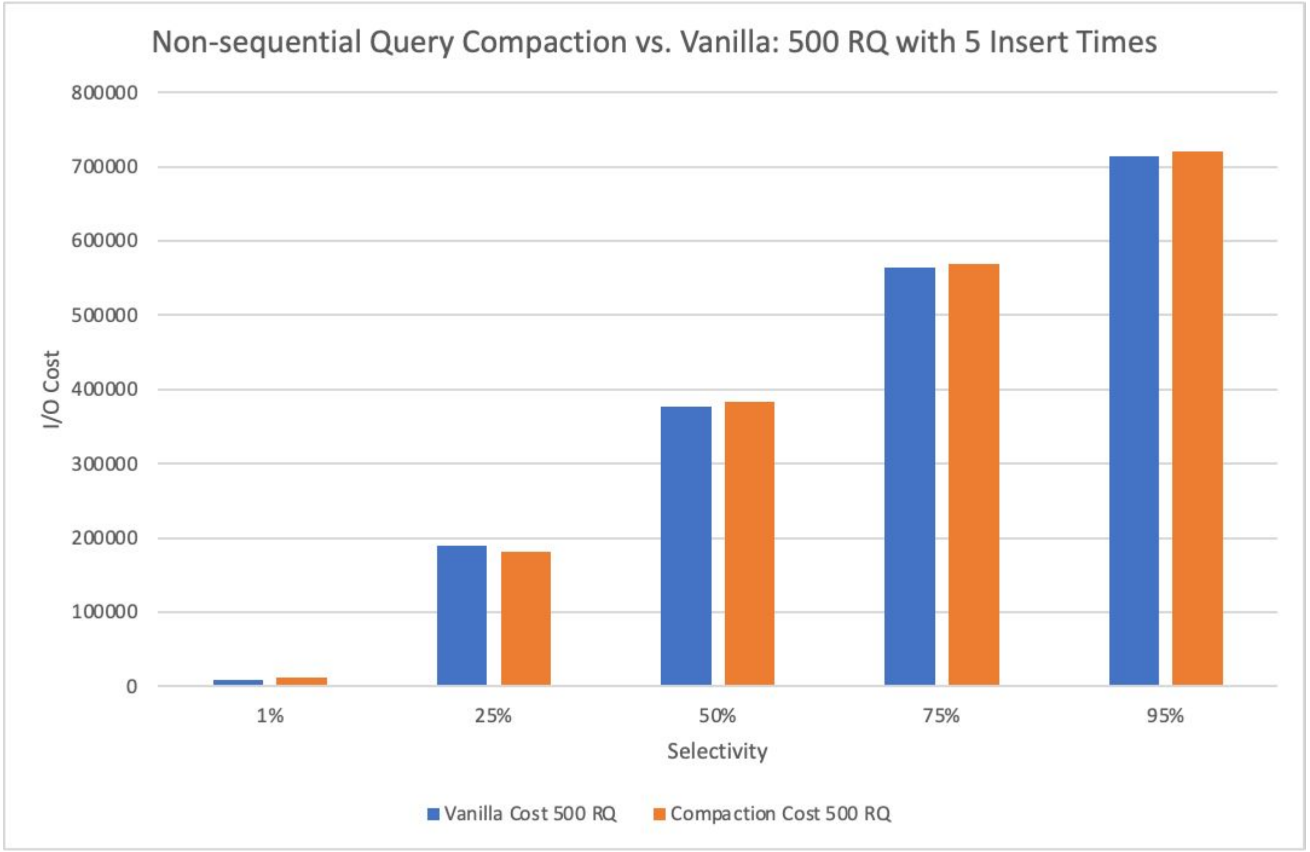
Non-sequential Query: 500 RQ, 5 times insert -75% Selectivity



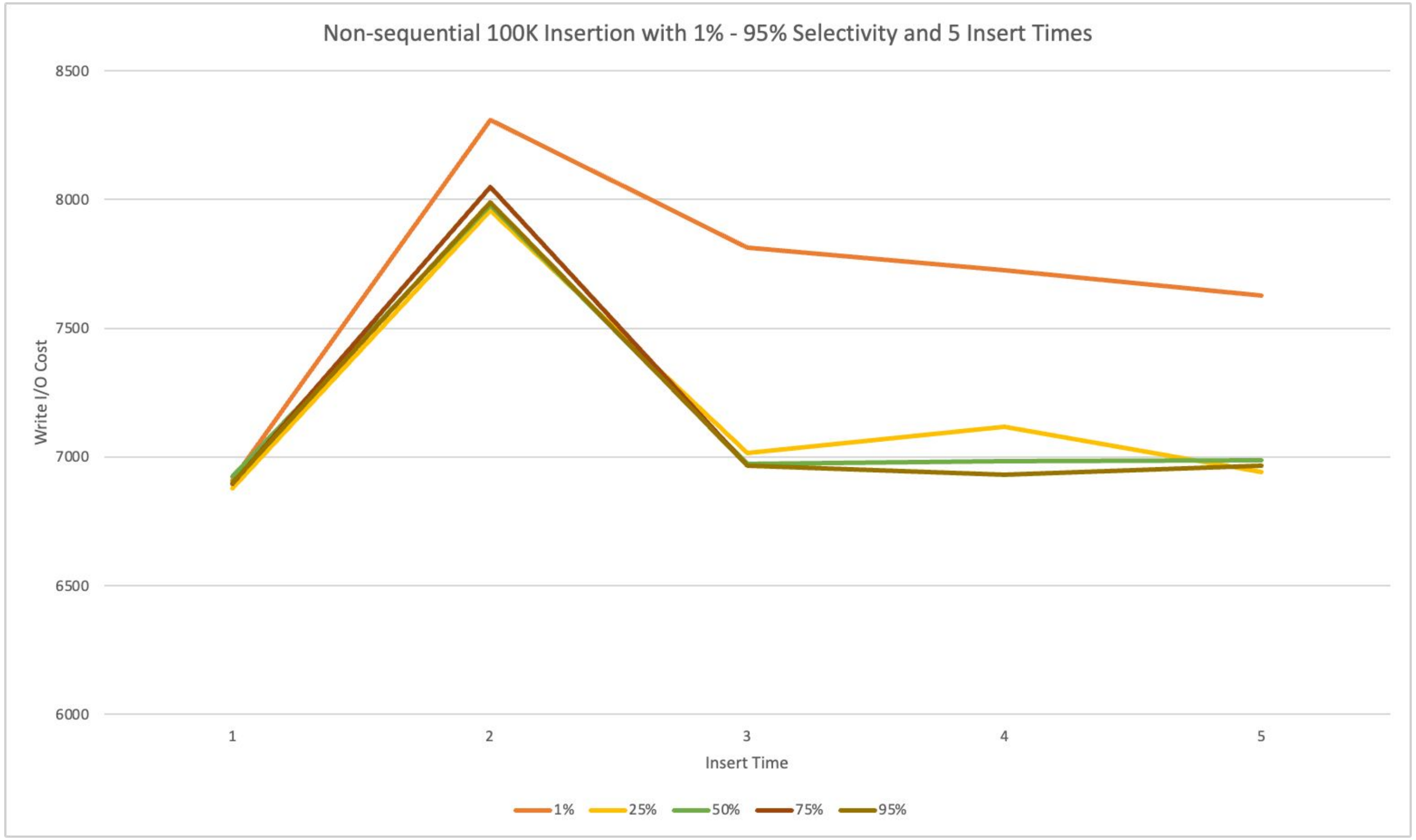
Non-sequential Query: 500 RQ, 5 times insert -95% Selectivity



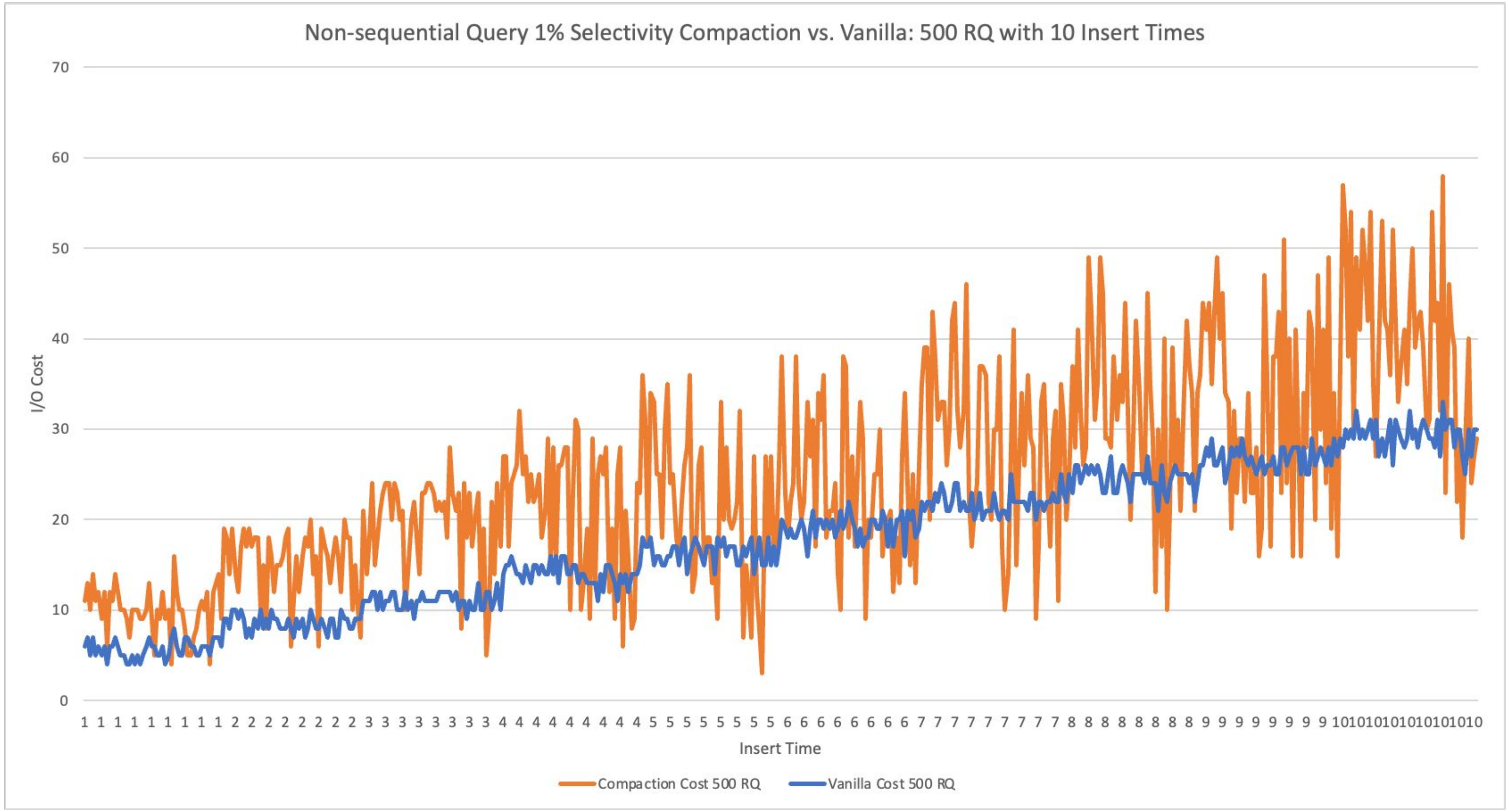
Non-sequential Query: 500 RQ, 5 times insert - Sum i/o



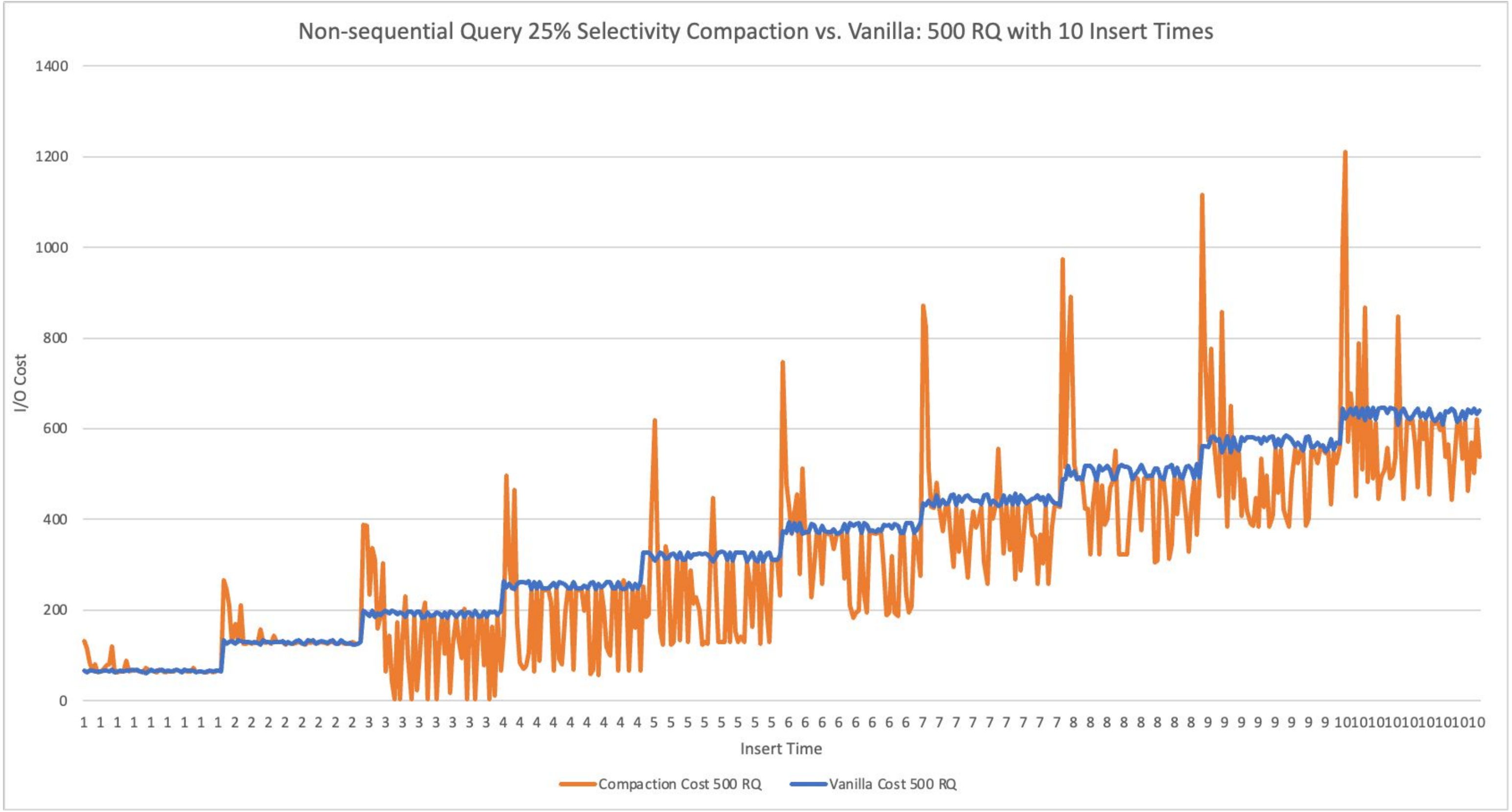
Non-sequential Query: 500 RQ, 5 times insert - Insertion



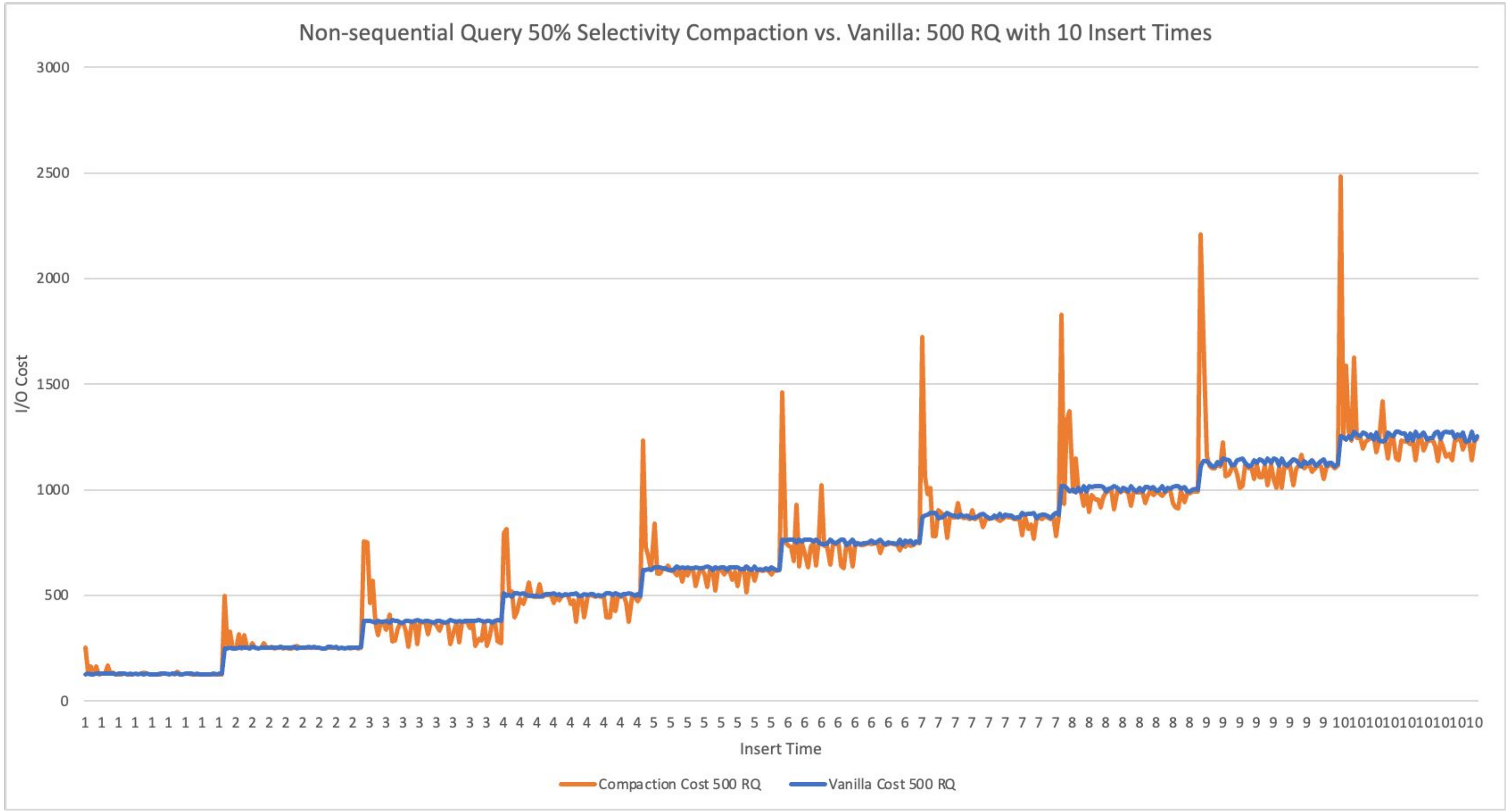
Non-sequential Query: 500 RQ, 10 times insert -1% Selectivity



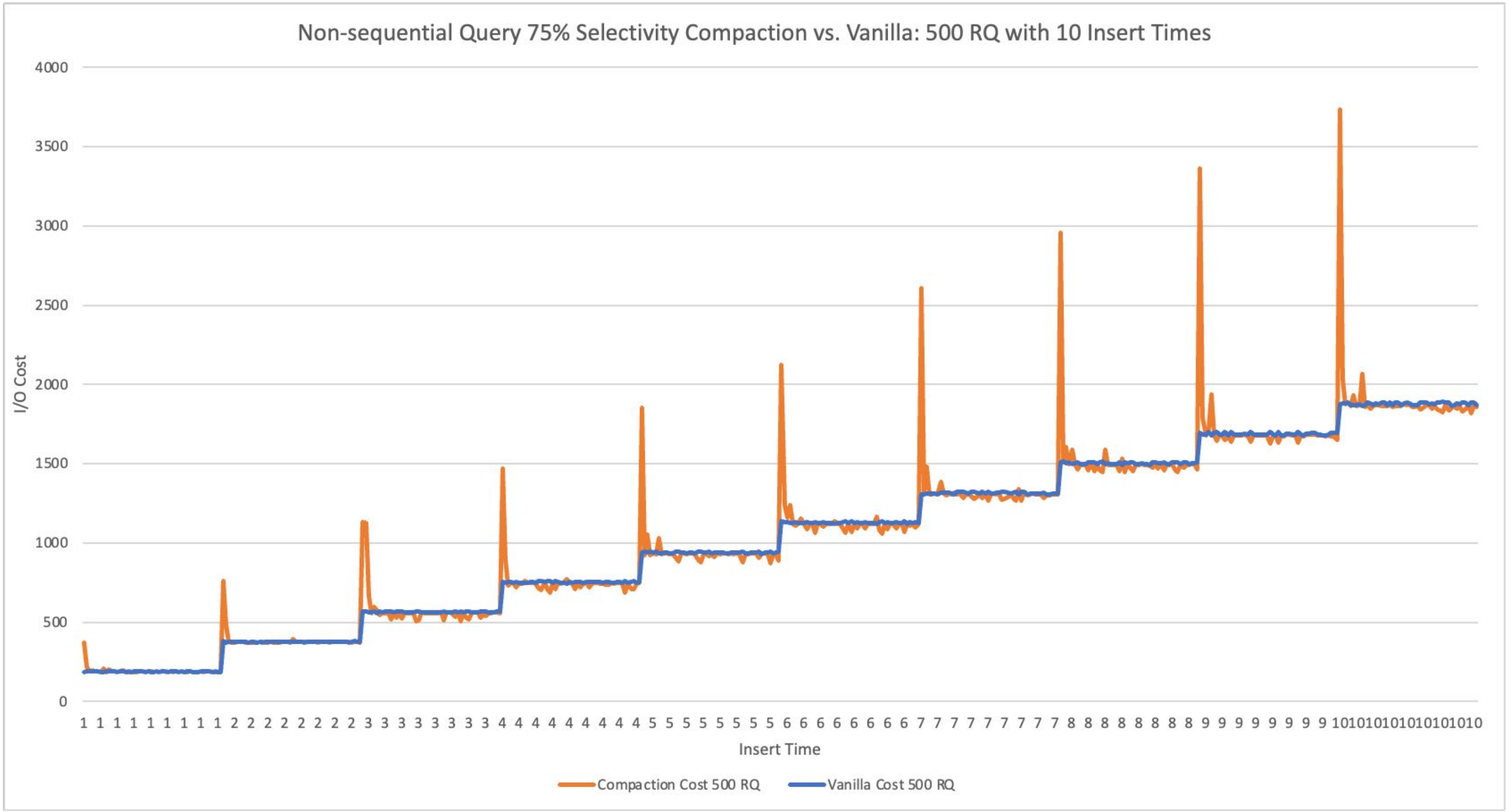
Non-sequential Query: 500 RQ, 10 times insert -25% Selectivity



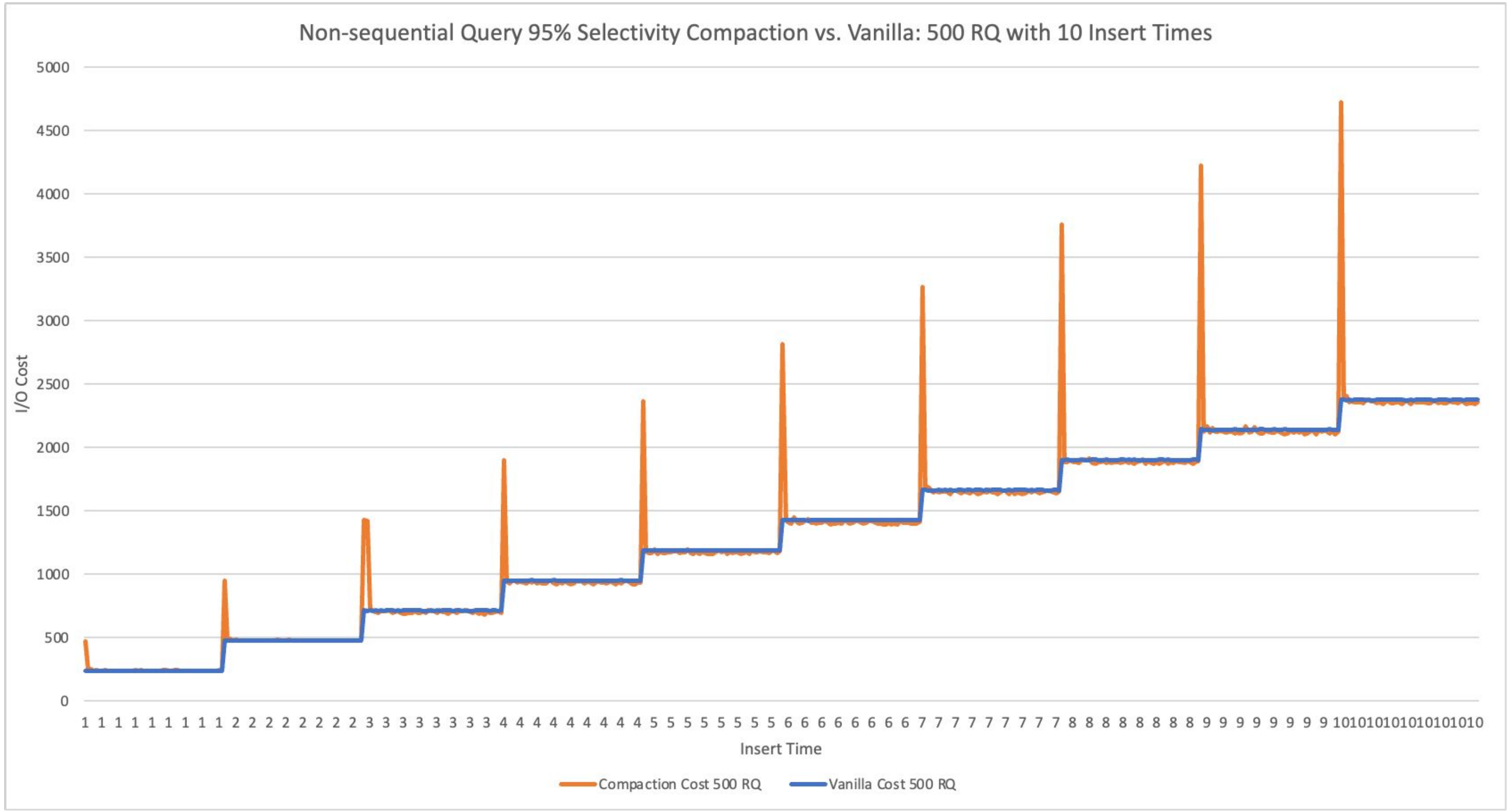
Non-sequential Query: 500 RQ, 10 times insert -50% Selectivity



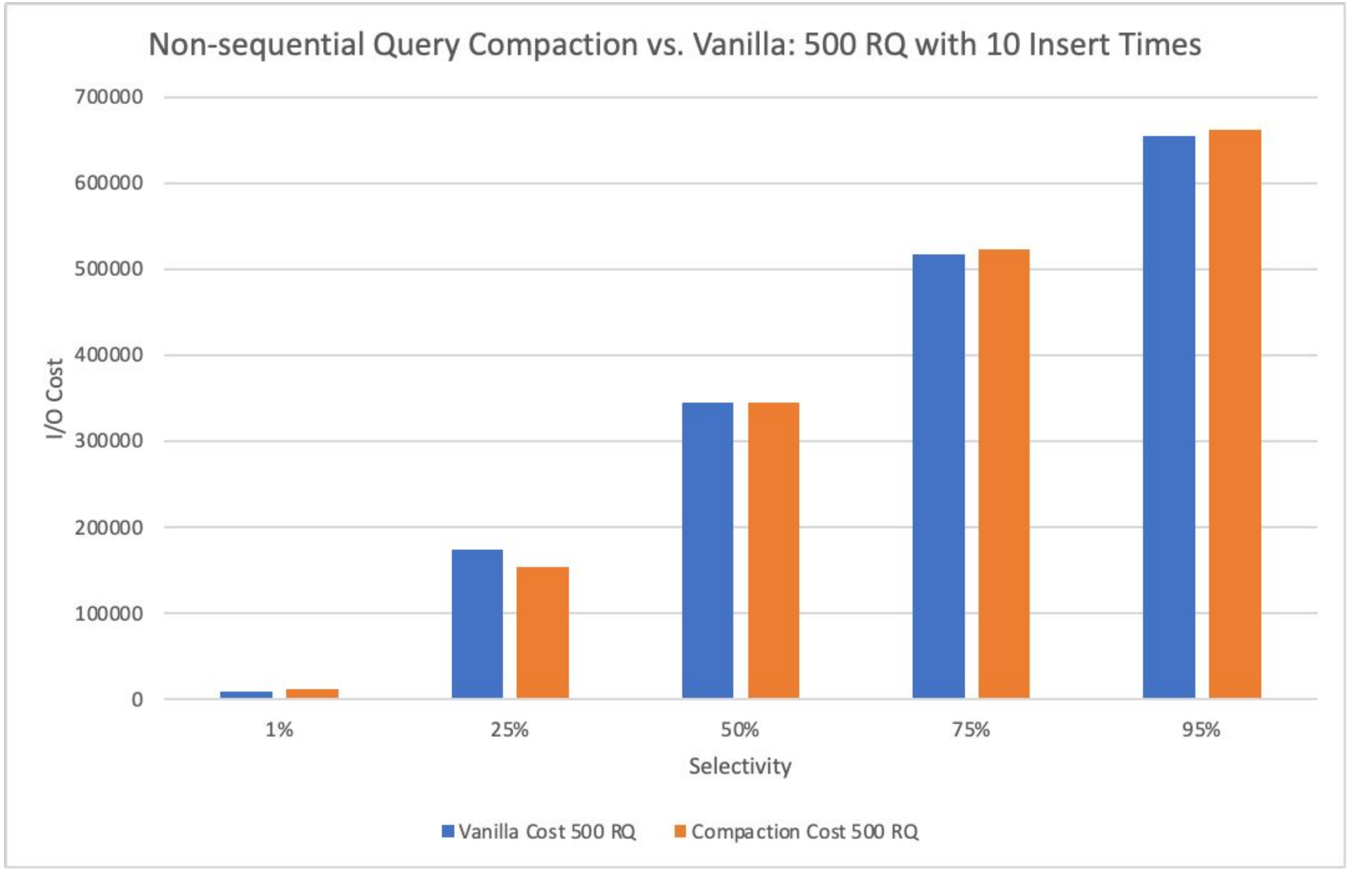
Non-sequential Query: 500 RQ, 10 times insert -75% Selectivity



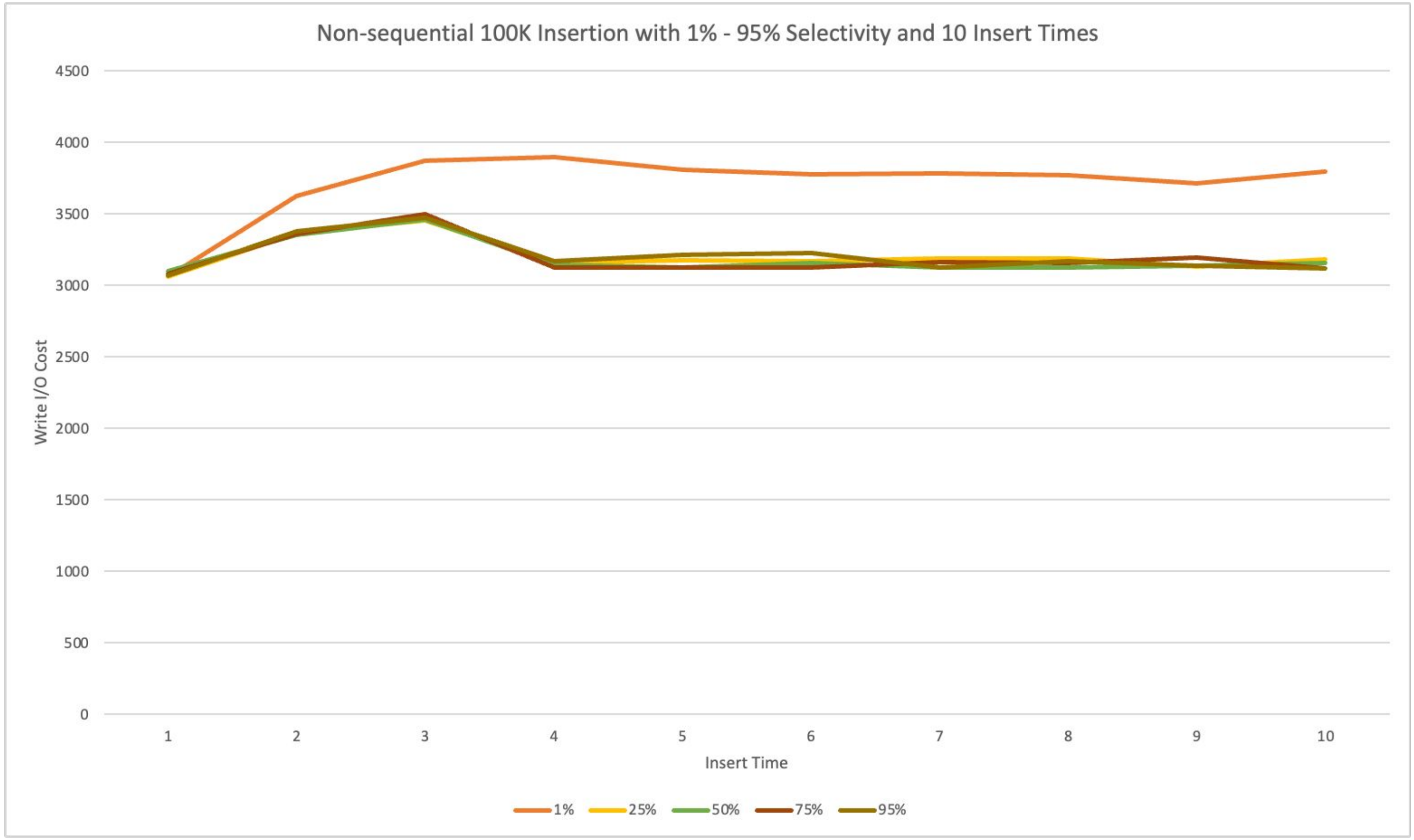
Non-sequential Query: 500 RQ, 10 times insert -95% Selectivity



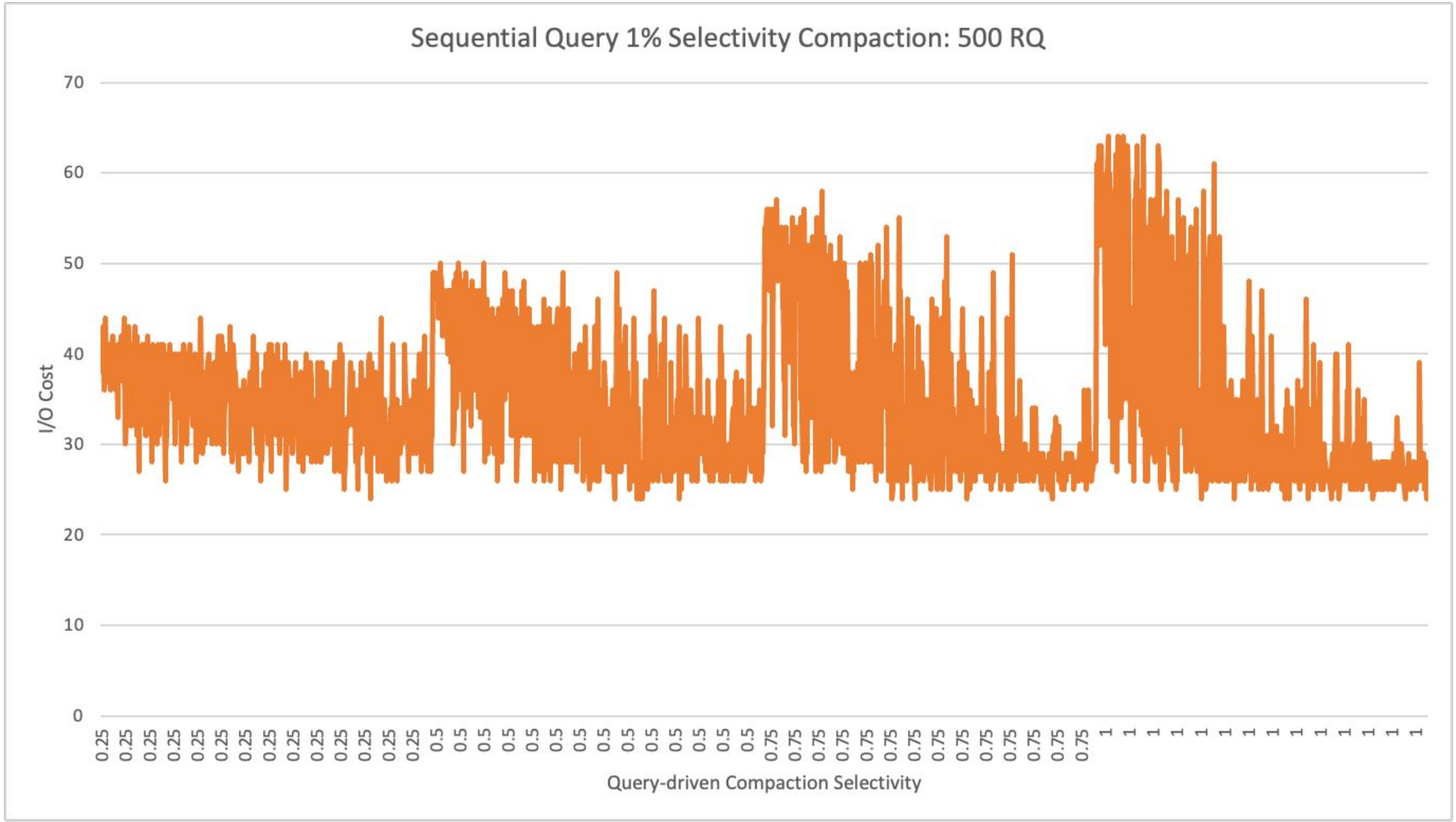
Non-sequential Query: 500 RQ, 10 times insert - Sum i/o



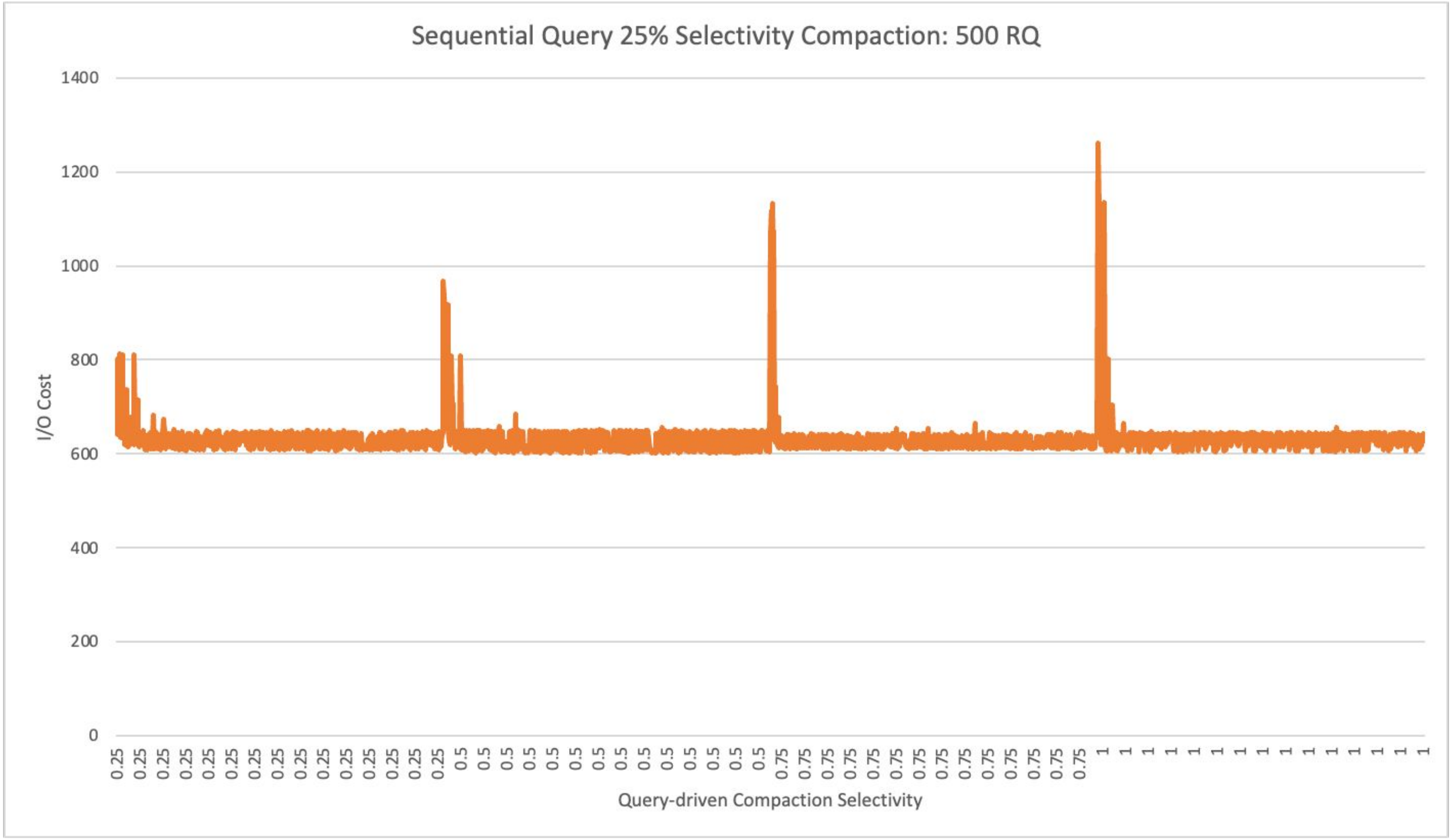
Non-sequential Query: 500 RQ, 10 times insert - Insertion



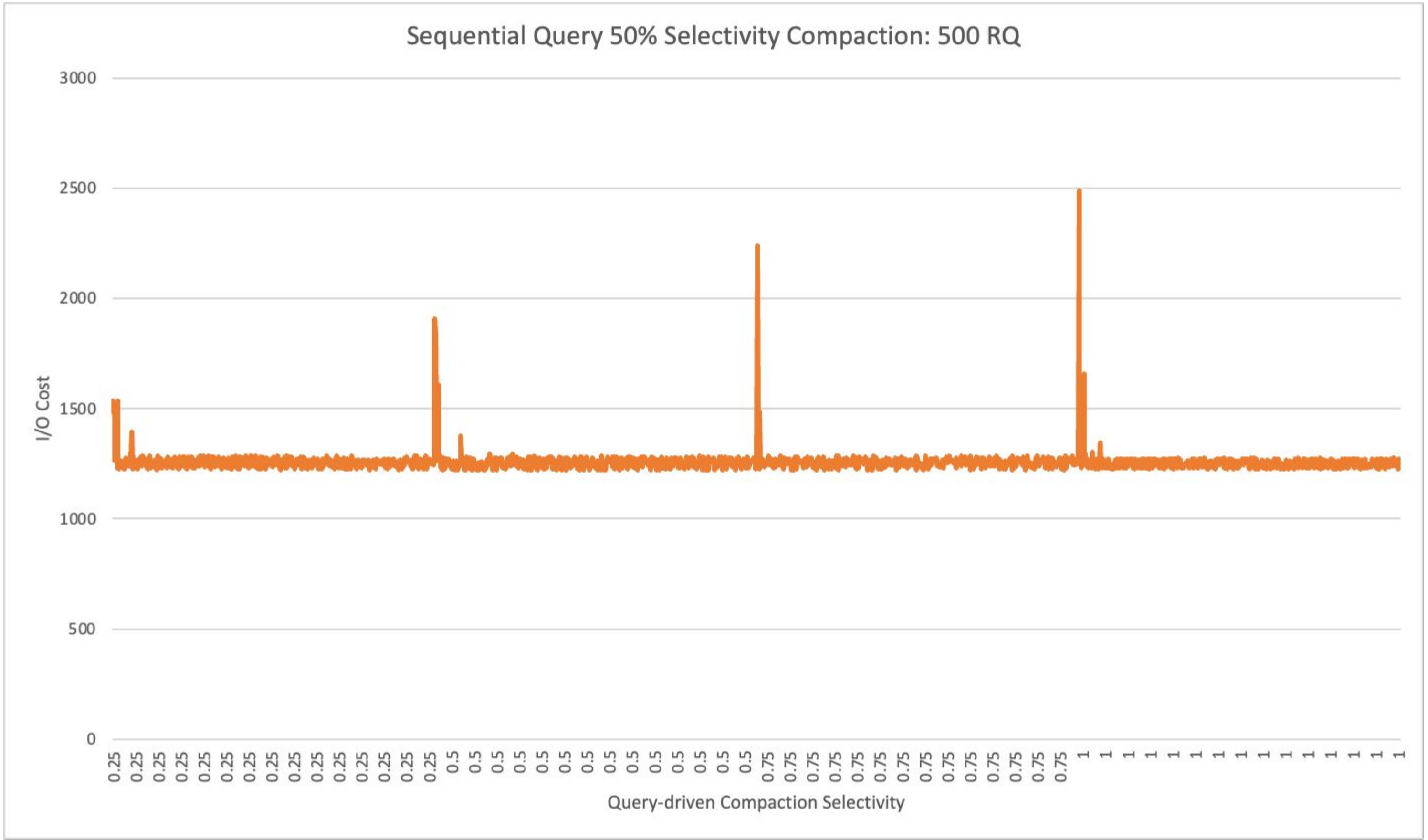
Compaction Selectivity: Sequential Query 1% - 500 RQ



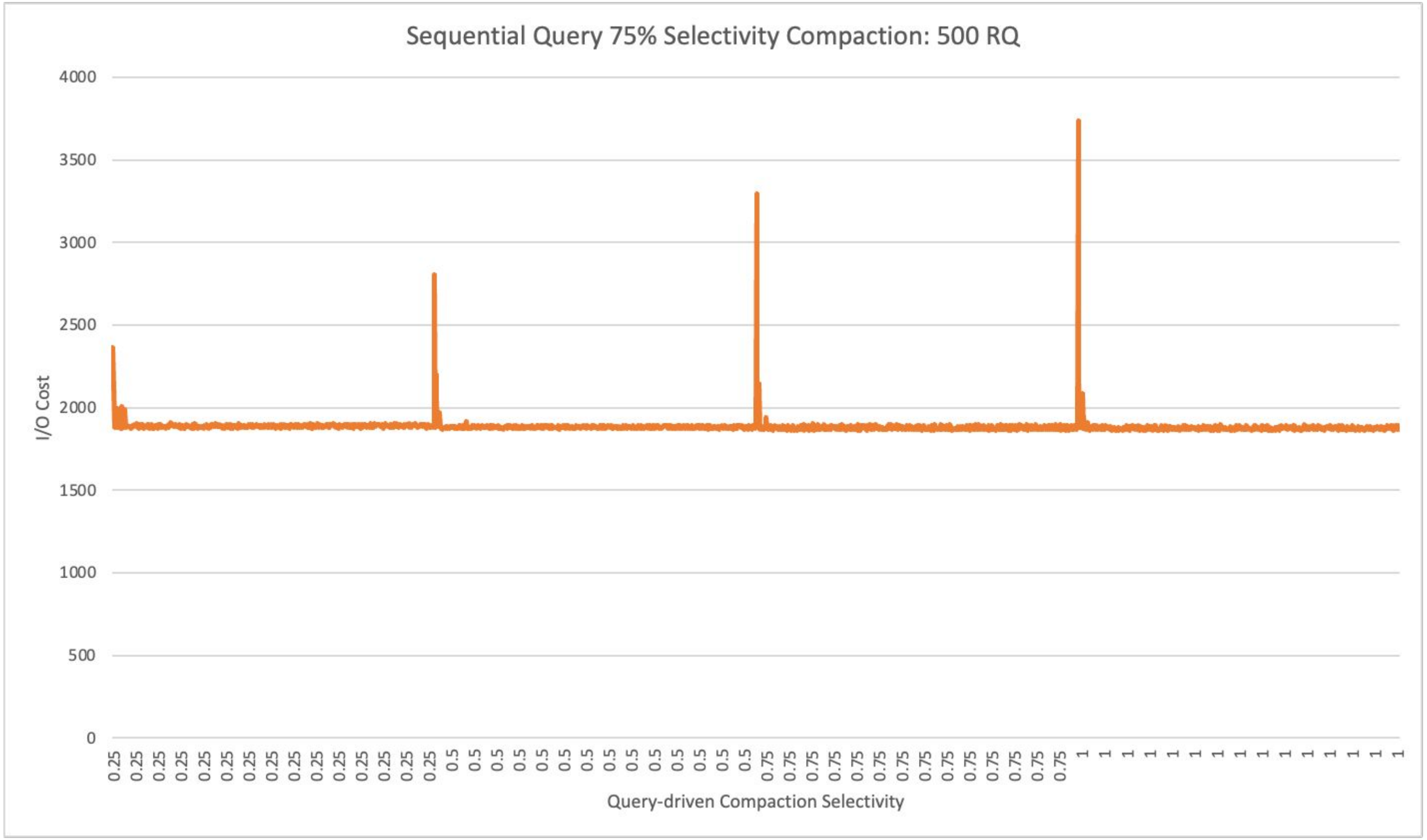
Compaction Selectivity: Sequential Query 25% - 500 RQ



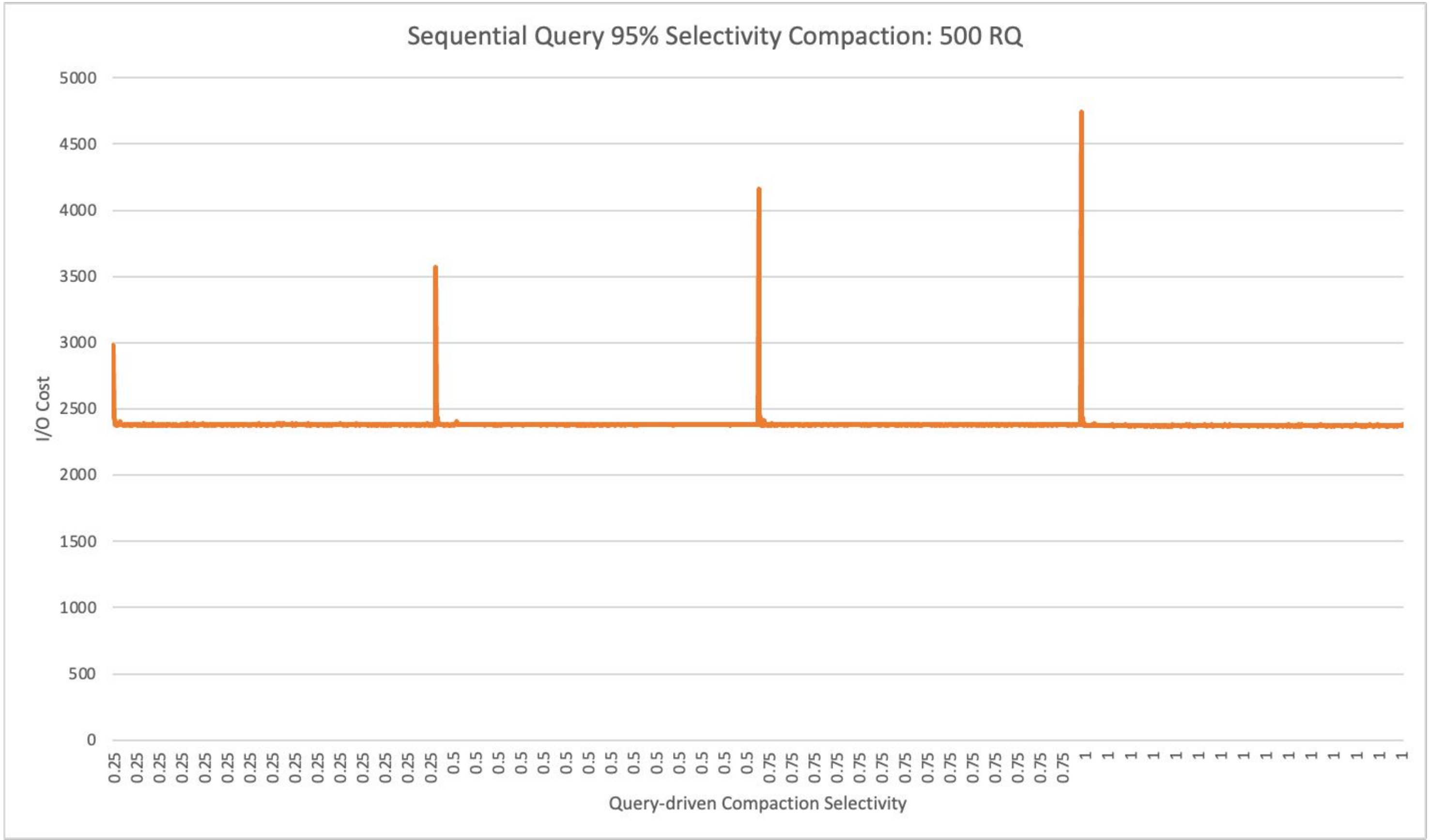
Compaction Selectivity: Sequential Query 50% - 500 RQ



Compaction Selectivity: Sequential Query 75% - 500 RQ



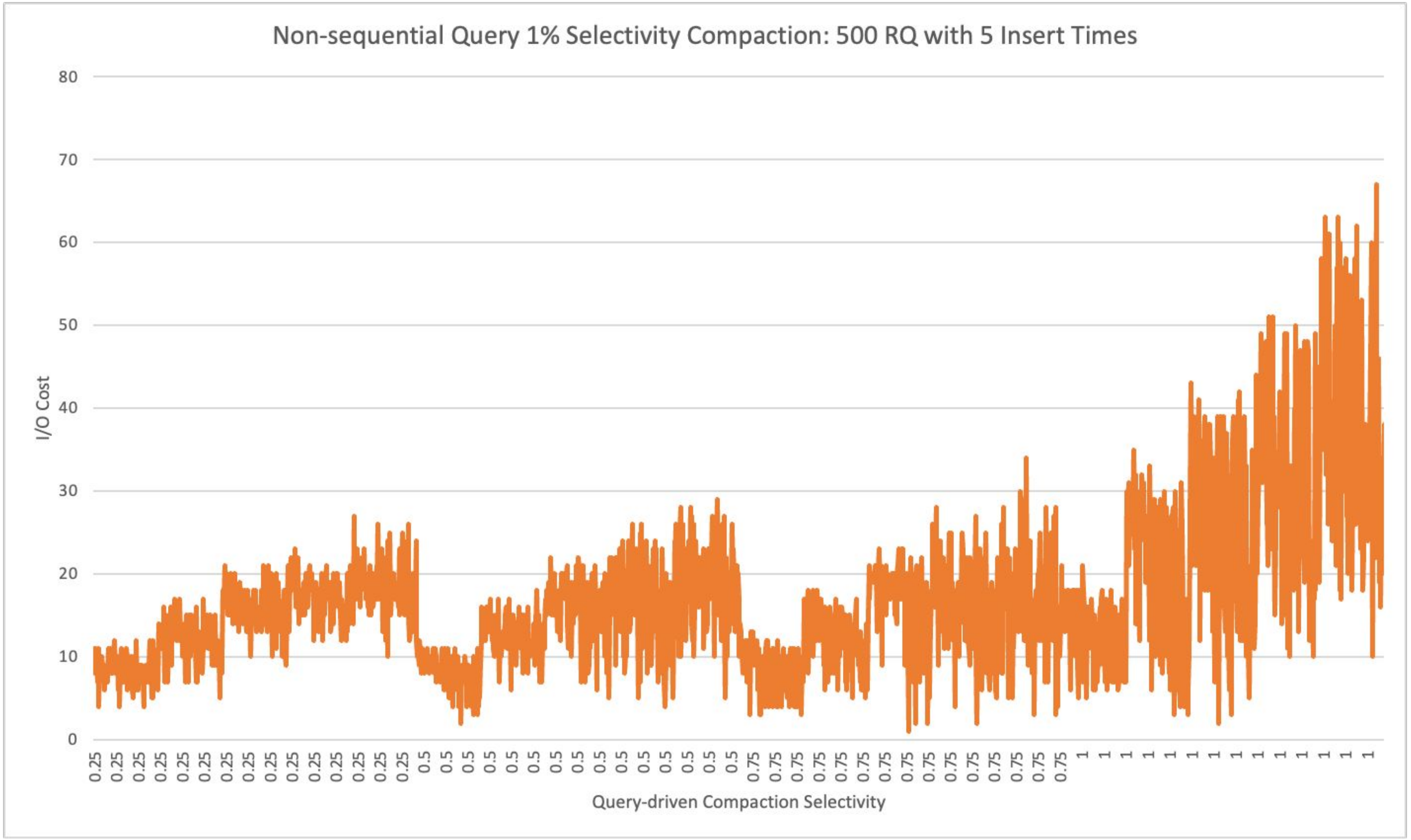
Compaction Selectivity: Sequential Query 95% - 500 RQ



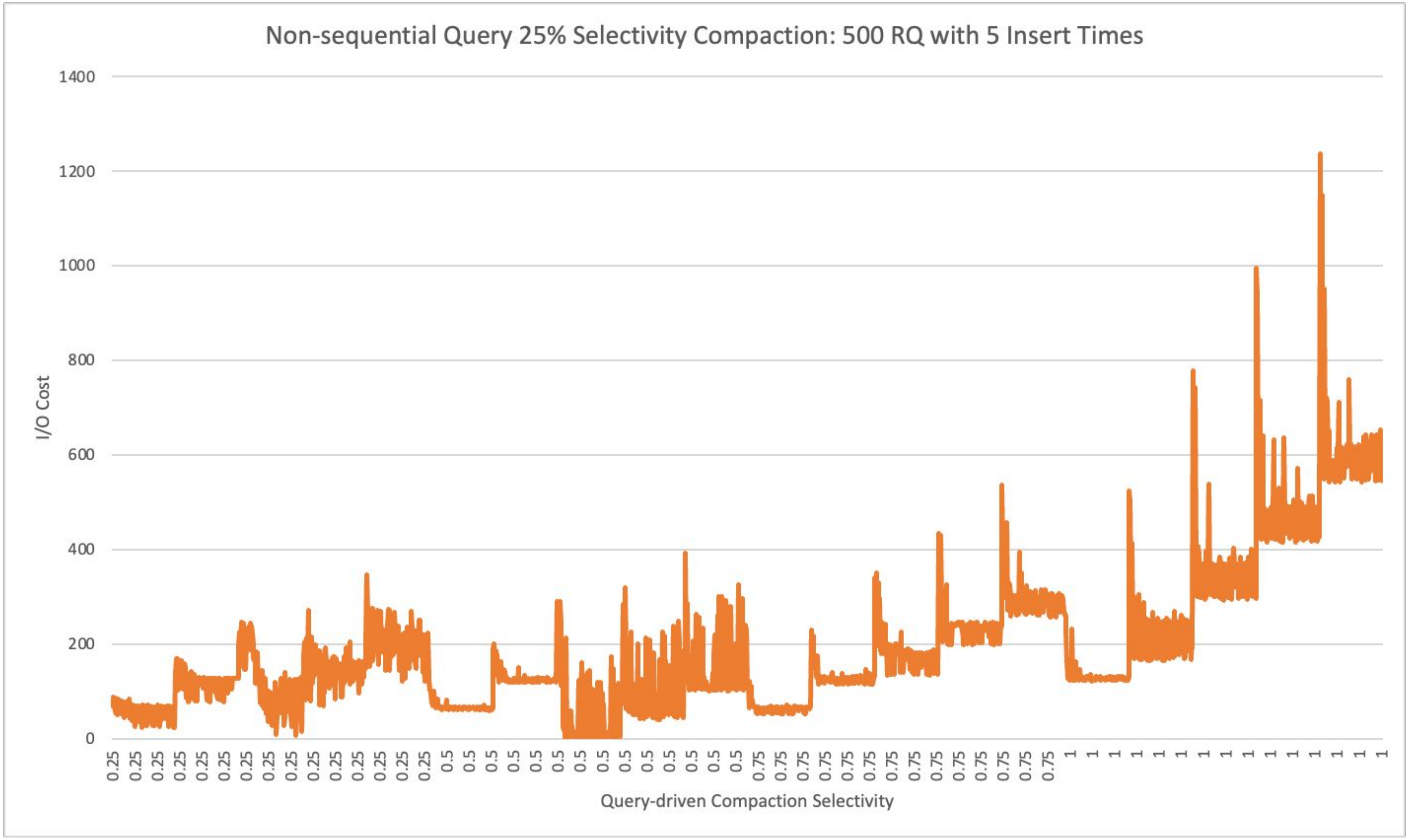
Compaction Selectivity: Sequential Query: 500 RQ - Sum i/o

No vanilla data

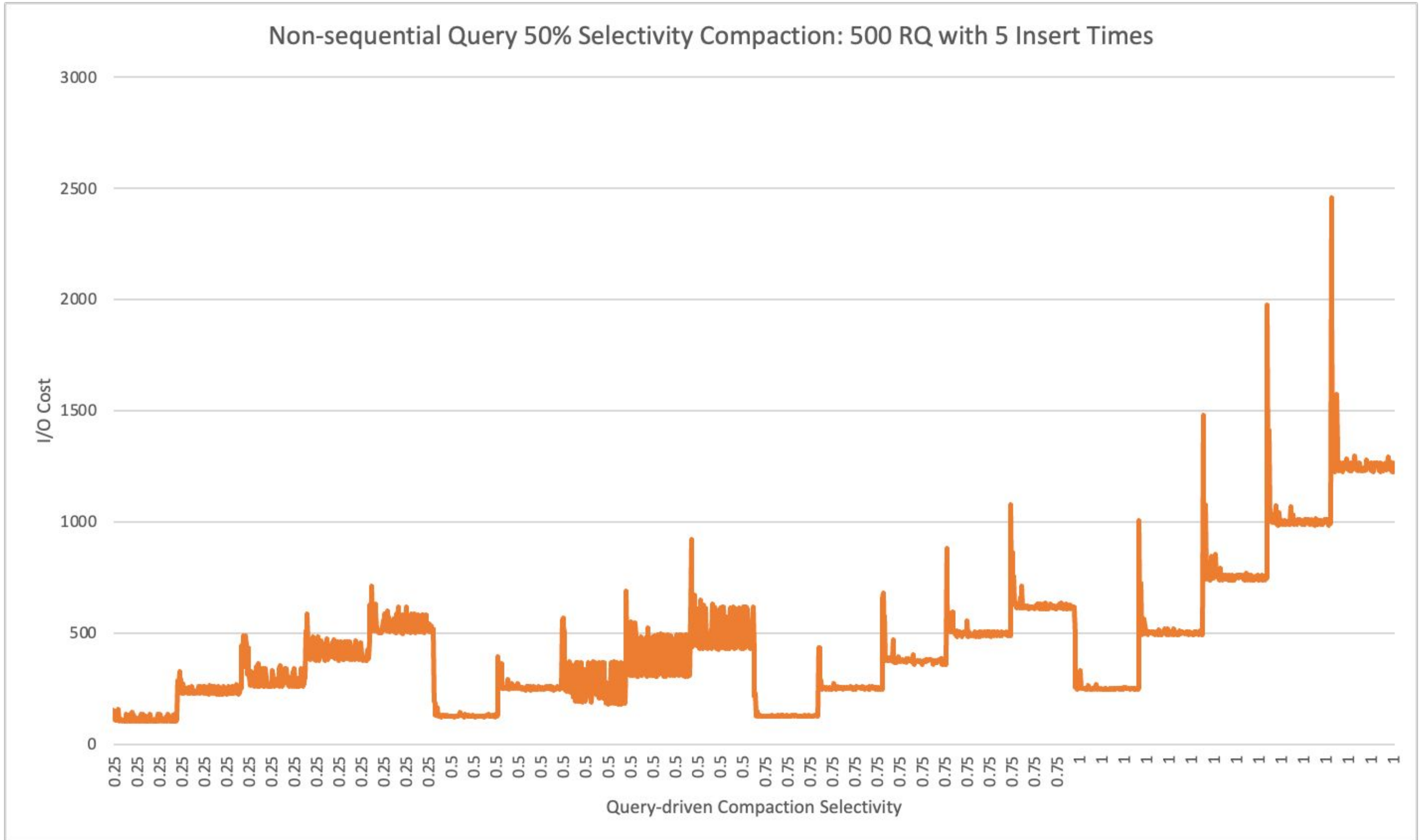
Compaction Selectivity: Non-sequential Query 1% - 500 RQ - 5 Times



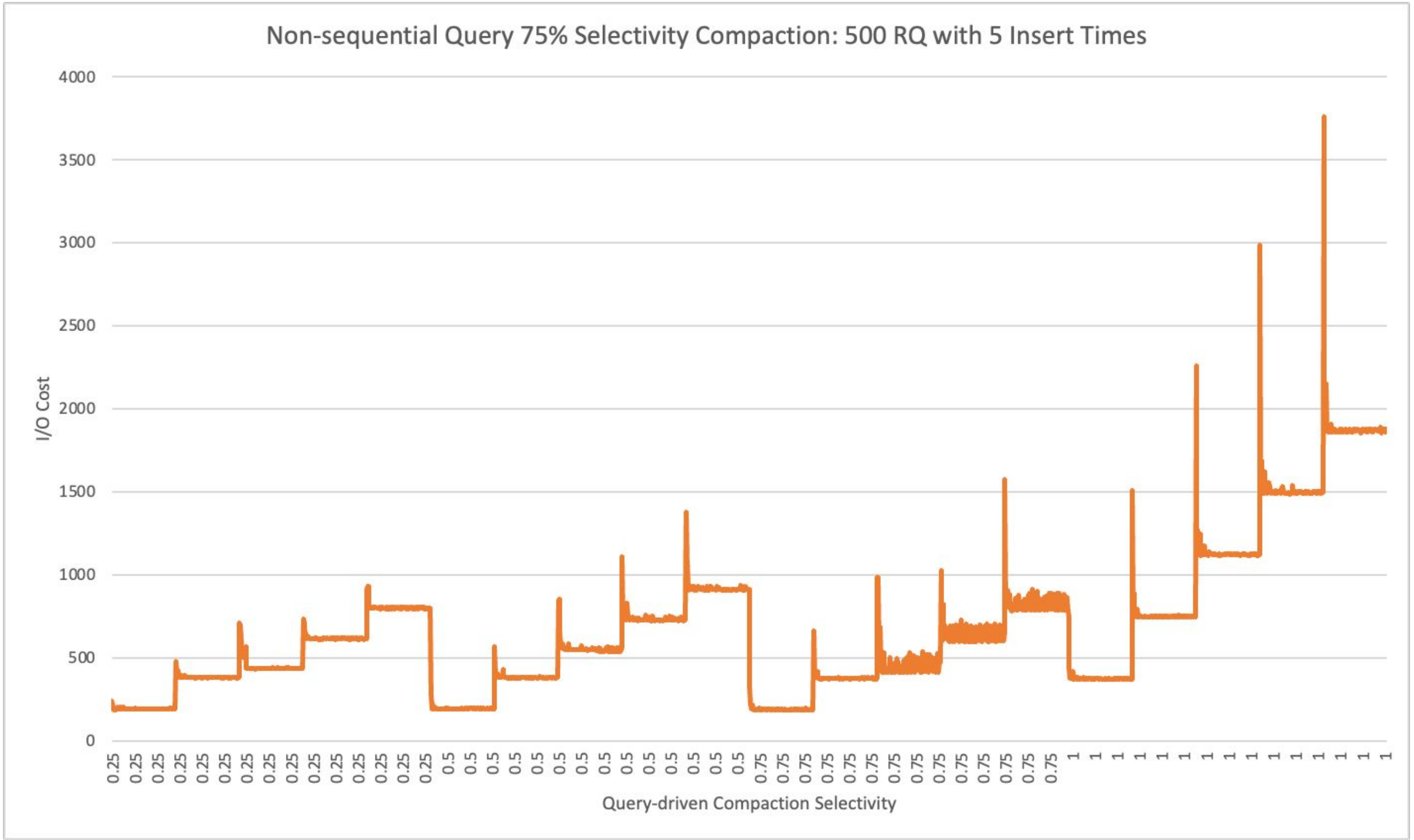
Compaction Selectivity: Non-sequential Query 25% - 500 RQ - 5 Times



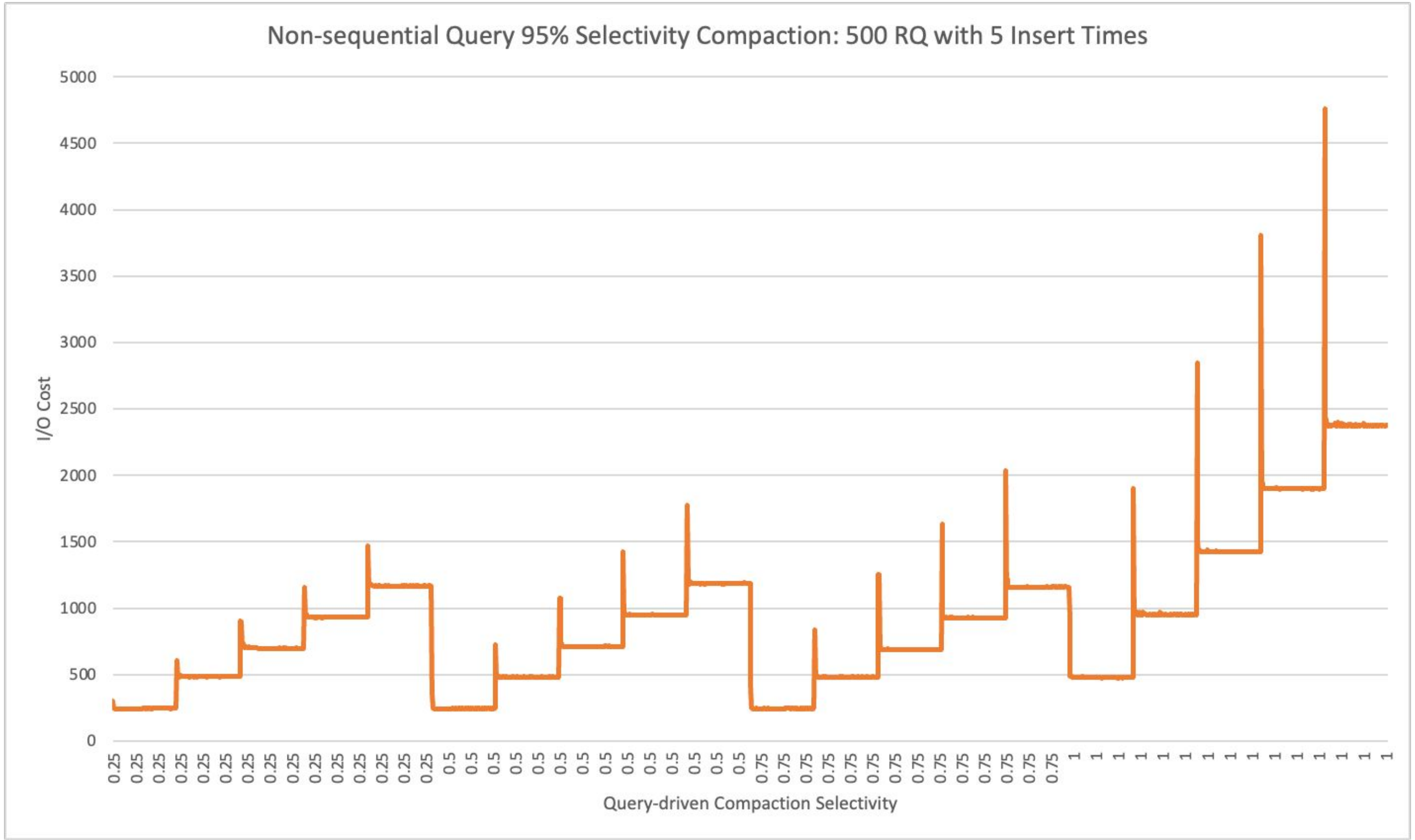
Compaction Selectivity: Non-sequential Query 50% - 500 RQ - 5 Times



Compaction Selectivity: Non-sequential Query 75% - 500 RQ - 5 Times

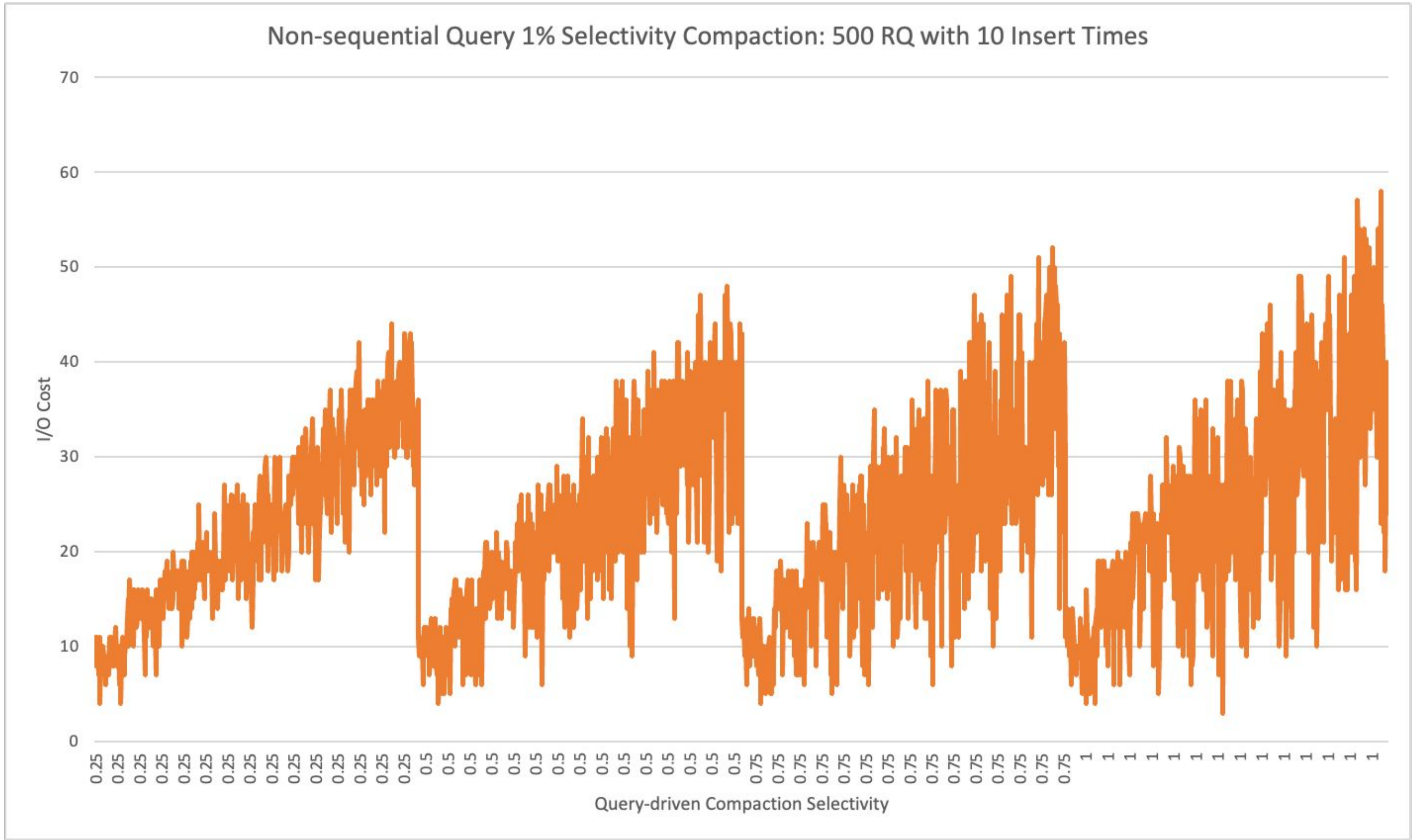


Compaction Selectivity: Non-sequential Query 95% - 500 RQ - 5 Times

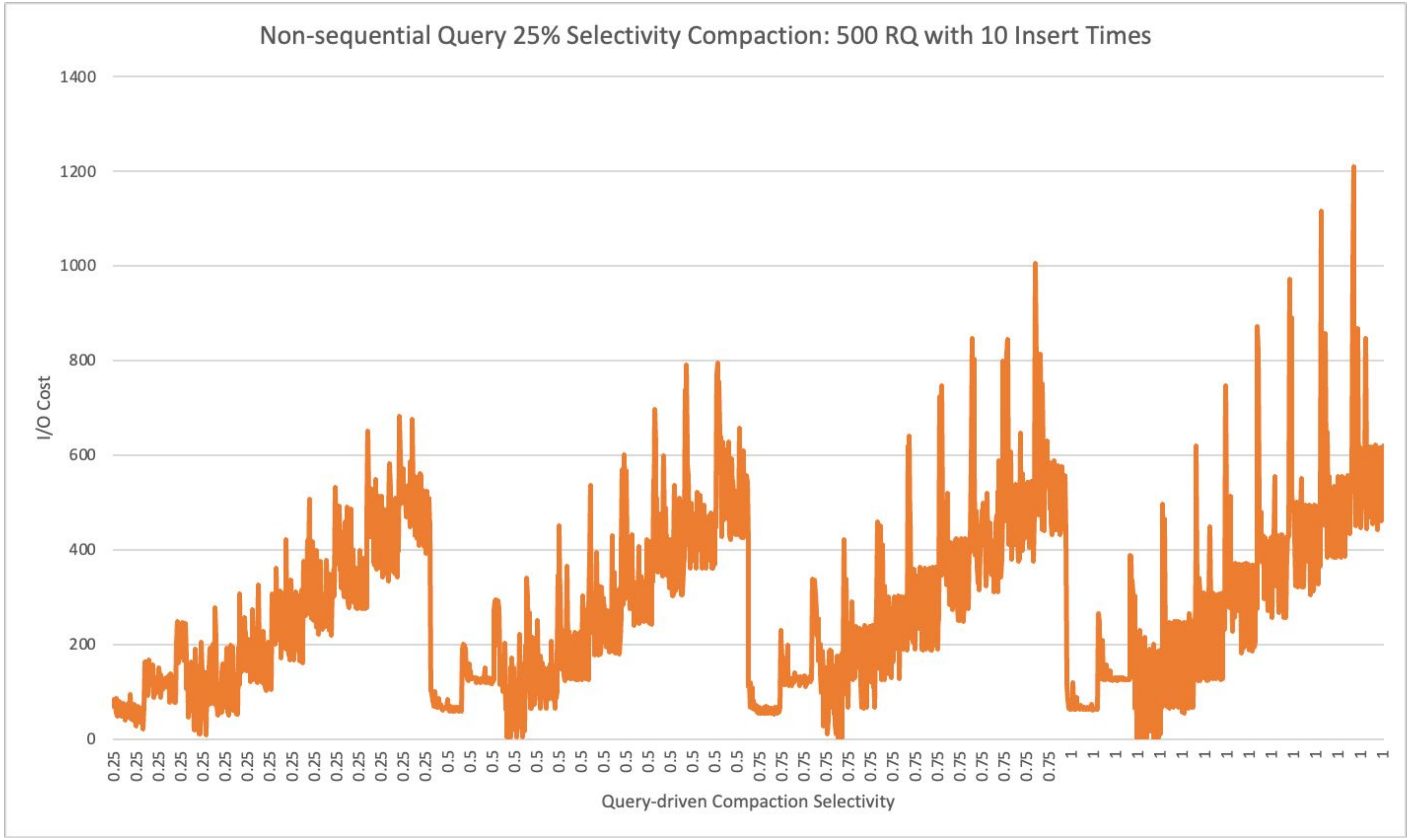


Compaction Selectivity: Non-sequential Query: 500 RQ - 5 Insert - Sum i/o

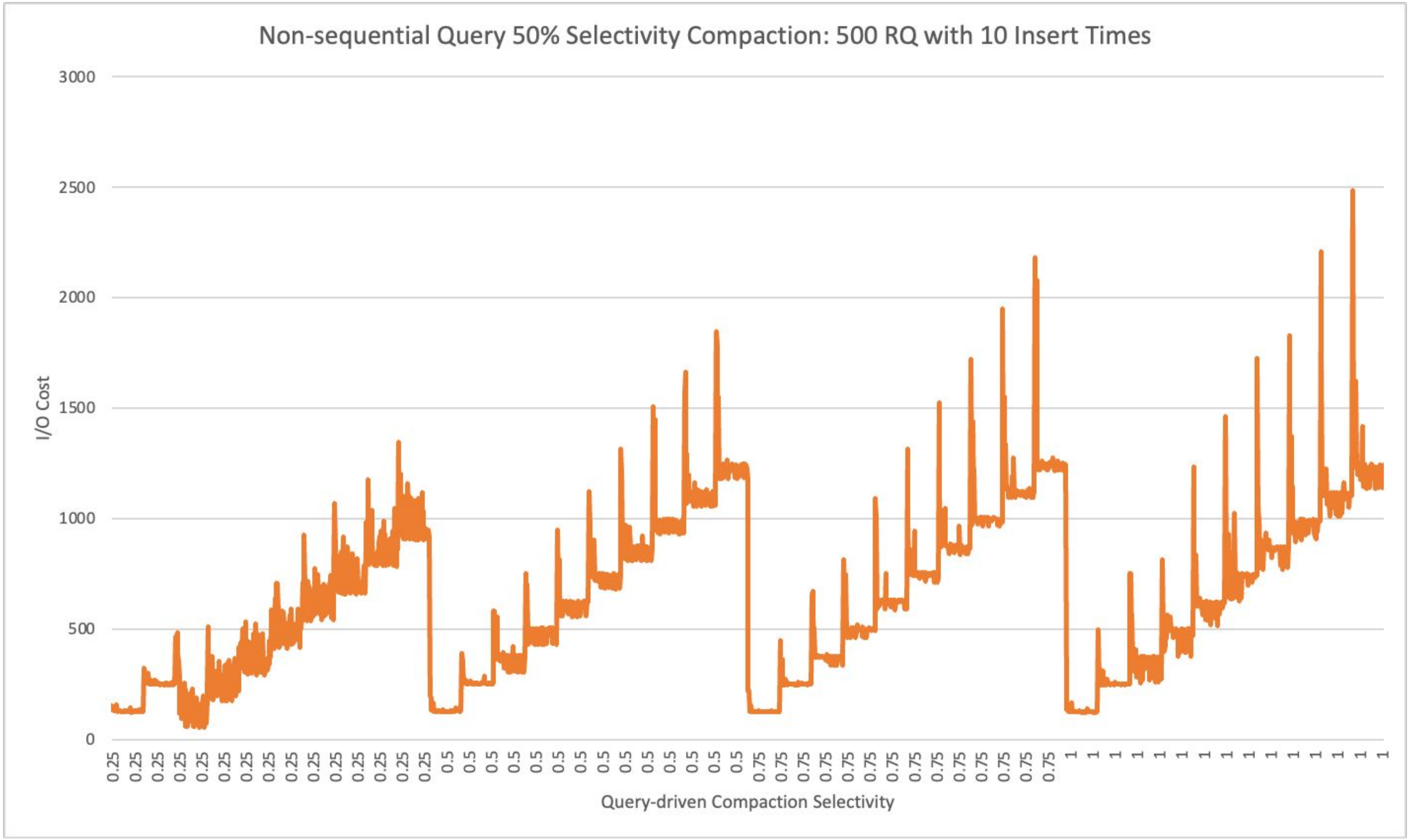
Compaction Selectivity: Non-sequential Query 1% - 500 RQ - 10 Times



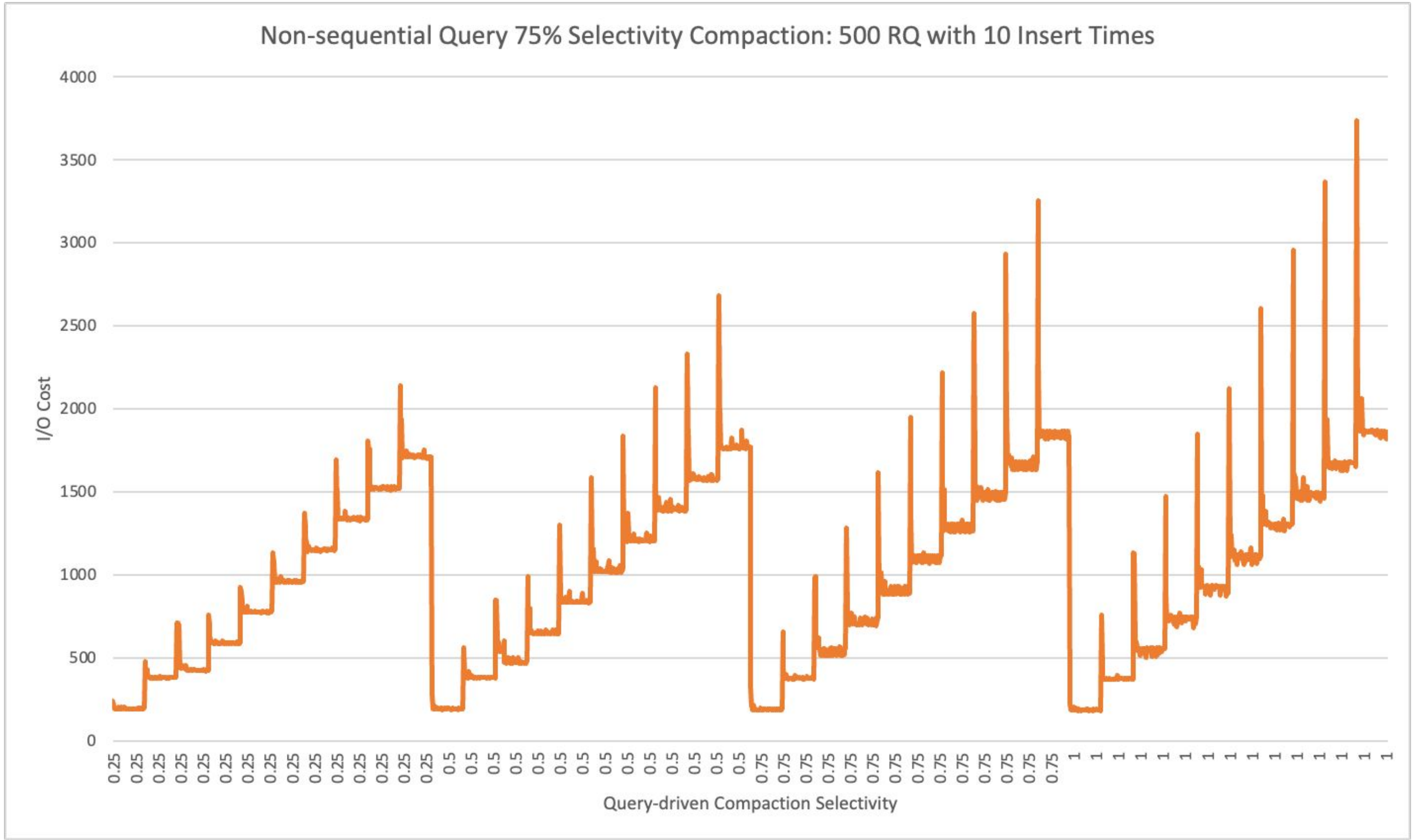
Compaction Selectivity: Non-sequential Query 25% - 500 RQ - 10 Times



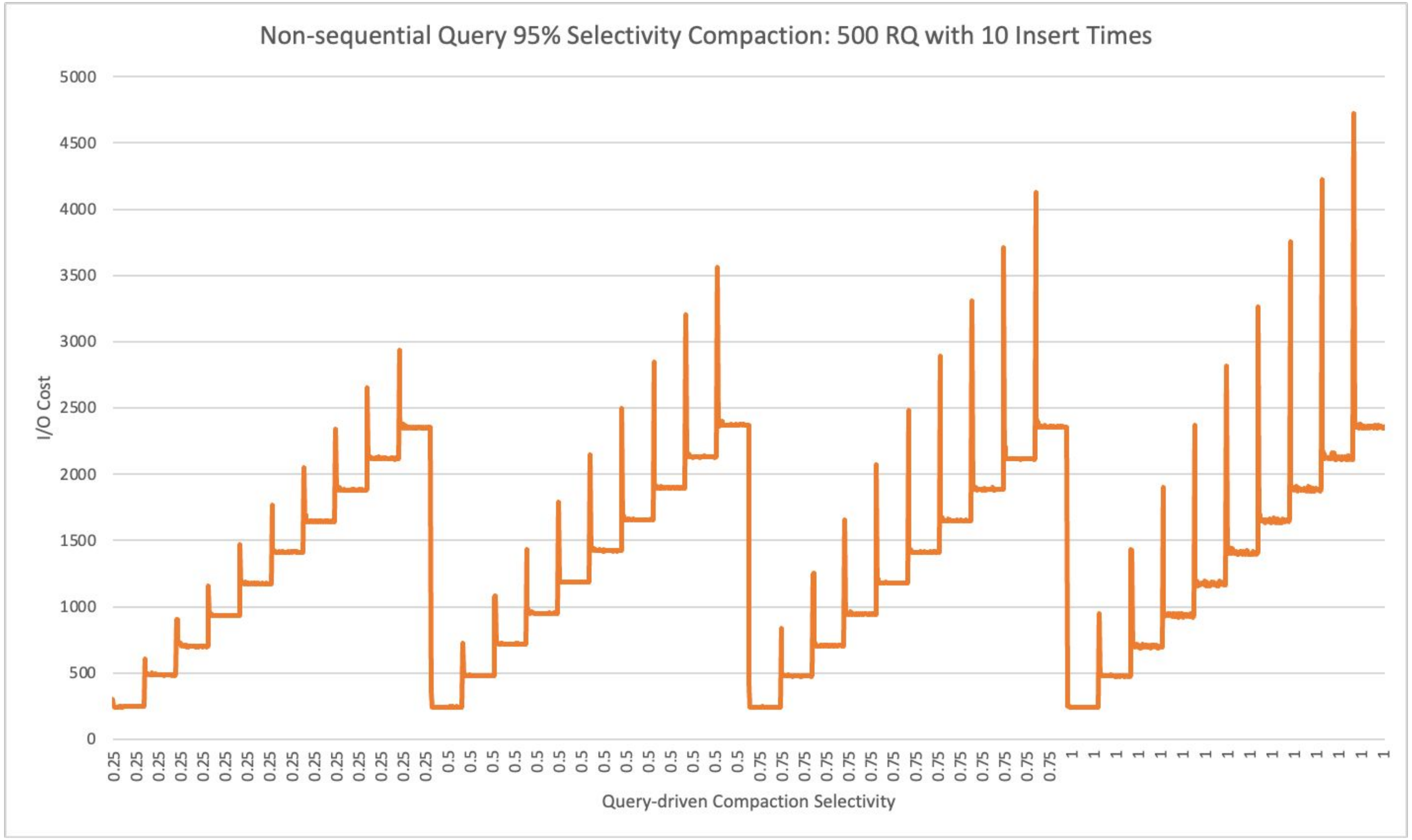
Compaction Selectivity: Non-sequential Query 50% - 500 RQ - 10 Times



Compaction Selectivity: Non-sequential Query 75% - 500 RQ - 10 Times



Compaction Selectivity: Non-sequential Query 95% - 500 RQ - 10 Times



Compaction Selectivity: Non-sequential Query: 500 RQ - 10 Insert - Sum i/o

No vanilla data

Sequential Query: 500 Point Query

No compaction data