Question 1

We would like to choose KV-store engine between BerkeleyDB vs. RocksDB for a workload consisting of 10% random gets and 90% random puts. BerkeleyDB uses a B-tree. RocksDB uses a leveled LSM-tree with size ratio T=10 and a buffer size P=2^{26} entries. Assume N=2^{40} and B=2^{7}. All internal nodes fit in memory. We're using a disk drive. What's your choice?

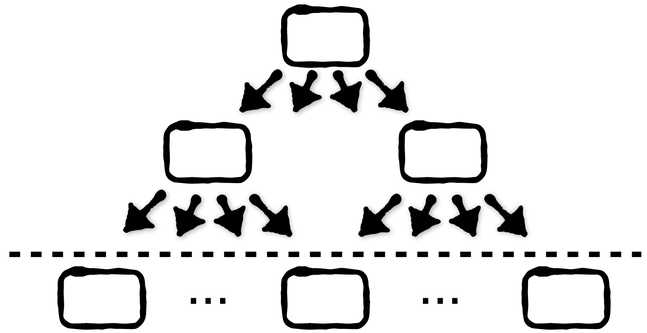
B-tree: Each "put" costs 1 read & 1 write I/O. Each get costs 1 read I/O.

As we are using disk, read/write I/O costs are symmetric.

Avg. #I/O per operation: 0.9 * 2 + 0.1 * 1 = 1.9

ORACLE

BERKELEY DB



Question 1

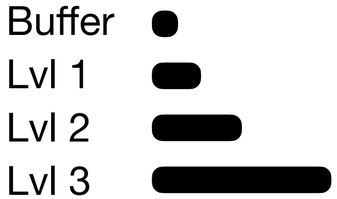
We would like to choose KV-store engine between BerkeleyDB vs. RocksDB for a workload consisting of 10% random gets and 90% random puts. BerkeleyDB uses a B-tree. RocksDB uses a leveled LSM-tree with size ratio T=10 and a buffer size $P=2^{26}$ entries. Assume $N=2^{40}$ and $B=2^{7}$. All internal nodes fit in memory. We're using a disk drive. What's your choice?

B-tree: Avg. #I/O per operation: 0.9 * 2 + 0.1 * 1 = 1.9

LSM-tree: A put costs $(T/B) * \log_T(N/P) = 0.32$ read & write I/Os

A get costs $\log_T(N/P) = 4.2$ read I/Os

0.32 * 2 * 0.9 + 4.2 * 0.1 = 0.996 (Cheaper)

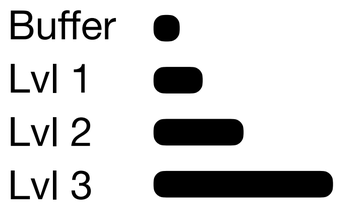


Question 2

A friend tells you they switched from using a B-tree to a basic LSM-tree (with size ratio 2), yet write-amplification actually increased. There are $N=2^{40}$ entries and $B=2^5$. Explain why this happened. Identify three tuning options for reducing write-amplification and the trade-off of each one of them.

LSM-tree before tuning: $(1/B) * \log_2(N/B) = 1.1$ write I/O (Costlier than B-tree)

Fact: increasing the buffer size reduces the number of levels and thus WA.

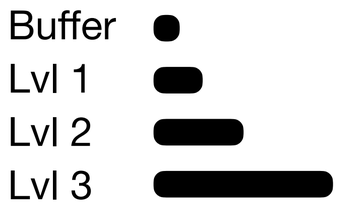


Question 2

A friend tells you they switched from using a B-tree to a basic LSM-tree (with size ratio 2), yet write-amplification actually increased. There are $N=2^{40}$ entries and $B=2^5$. Explain why this happened. Identify three tuning options for reducing write-amplification and the trade-off of each one of them.

LSM-tree before tuning: $(1/B) * \log_2(N/B) = 1.1$ write I/O (Costlier than B-tree)

Fact: increasing the buffer size reduces the number of levels and thus WA.



Let the buffer size be P entries. We can increase the buffer size to decrease the number of levels over which entries get merged. E.g., P=2^12 entries. Trade-off: requires more memory.

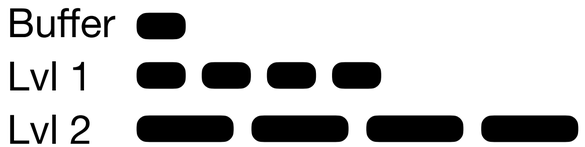
$$(1/B) * \log_2(N/P) = 0.87$$

Question 2

A friend tells you they switched from using a B-tree to a basic LSM-tree (with size ratio 2), yet write-amplification actually increased. There are $N=2^{40}$ entries and $B=2^5$. Explain why this happened. Identify three tuning options for reducing write-amplification and the trade-off of each one of them.

We can further employ tiering, say, with size ratio T=10. Trade-off: reads get more expensive.

e.g., (1/B) * log_T(N/P) = 0.28

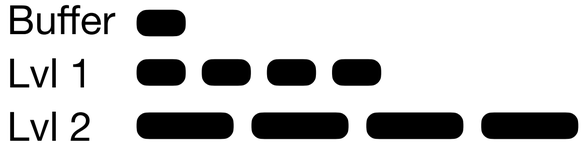


Question 2

A friend tells you they switched from using a B-tree to a basic LSM-tree (with size ratio 2), yet write-amplification actually increased. There are $N=2^{40}$ entries and $B=2^5$. Explain why this happened. Identify three tuning options for reducing write-amplification and the trade-off of each one of them.

We can further employ tiering, say, with size ratio T=10. Trade-off: reads get more expensive.

e.g., (1/B) * log_T(N/P) = 0.28



Last approach: Make the LSM-tree unclustered. More in next question.

Question 3

In a clustered LSM-trees, the values are stored within the LSM-tree alongside their keys. Another approach is an unclustered LSM-tree, which stores values in an append-only file and indexes them using key-pointer pairs from within the LSM-tree. What's the impact of clustered vs. unclustered LSM-trees on put/get/scan performance. Propose adjusted cost models assuming a basic LSM-tree (size ratio T=2). Hint: let K be the number of key-pointer pairs fitting into a page, and let B be the number of key-value pairs fitting in a page. Based on your models, identify cases where each of these approaches shines. Assume a 1 page buffer.

Question 3

In a clustered LSM-trees, the values are stored within the LSM-tree alongside their keys. Another approach is an unclustered LSM-tree, which stores values in an append-only file and indexes them using key-pointer pairs from within the LSM-tree. What's the impact of clustered vs. unclustered LSM-trees on put/get/scan performance. Propose adjusted cost models assuming a basic LSM-tree (size ratio T=2). Hint: let K be the number of key-pointer pairs fitting into a page, and let B be the number of key-value pairs fitting in a page. Based on your models, identify cases where each of these approaches shines. Assume a 1 page buffer.

$L=\log_T(N/B)$

Clustered, put: O(L / B)).

Clustered, get: L.

Clustered, scan: O(L+S/B)

$L=\log_T(N/K)$

Unclustered, put: O(1/B + L/K)).

Unclustered, get: L + 1.

Unclustered, scan: O(L+S)

Question 3

In a clustered LSM-trees, the values are stored within the LSM-tree alongside their keys. Another approach is an unclustered LSM-tree, which stores values in an append-only file and indexes them using key-pointer pairs from within the LSM-tree. What's the impact of clustered vs. unclustered LSM-trees on put/get/scan performance. Propose adjusted cost models assuming a basic LSM-tree (size ratio T=2). Hint: let K be the number of key-pointer pairs fitting into a page, and let B be the number of key-value pairs fitting in a page. Based on your models, identify cases where each of these approaches shines. Assume a 1 page buffer.

$L=\log_T(N/B)$

Clustered, put: O(L / B)).

Clustered, get: L.

Clustered, scan: O(L+S/B)

$L=\log_T(N/K)$

Unclustered, put: O(1/B + L/K)).

Unclustered, get: L + 1.

Unclustered, scan: O(L+S)

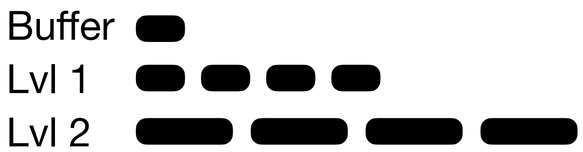
For small entries, clustered is better.

For large entries, unclustered is better.

Question 4

An LSM-tree can store multiple versions for a given entry, where only the most recent is considered up-to-date and the rest are obsolete. The obsolete entries are eventually discarded during compaction, but meanwhile they consume space. This phenomenon is known as space-amplification, defined as: (physical space taken up) / (logical data size). Quantify space-worst-case amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full.

Tiered



Leveled

Question 4

An LSM-tree can store multiple versions for a given entry, where only the most recent is considered up-to-date and the rest are obsolete. The obsolete entries are eventually discarded during compaction, but meanwhile they consume space. This phenomenon is known as space-amplification, defined as: (physical space taken up) / (logical data size). Quantify space-worst-case amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full.

Tiered

≈T

Leveled

≈1+1/(T-1)