Tutorial on LSM-trees

Database System Technology

Niv Dayan

Groups

Form for registration was sent today :)
Complete by Sunday

Question 1 - picking a storage engine

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 of $P=2^{26}$ entries. Assume $N=2^{40}$ and $B=2^{7}$. All internal nodes fit in memory. We are using a disk drive. What is your choice and why?

ORACLE
BERKELEY DB

RocksDB

Question 1 - picking a storage engine

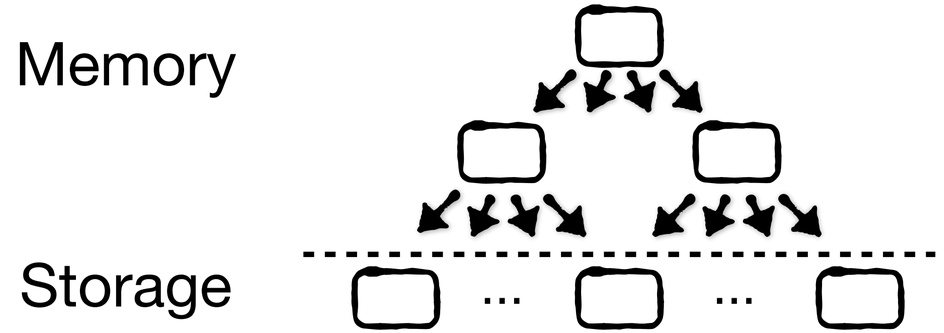
We would like to choose KV-store engine between BerkeleyDB vs. RocksDB for a workload consisting of 10% random gets and 90% random puts. BerkeleyDB uses a B-tree. RocksDB uses a leveled LSM-tree with size ratio T=10 and a buffer size of $P=2^{26}$ entries. Assume $N=2^{40}$ and $B=2^{7}$. All internal nodes fit in memory. We are using a disk drive. What is your choice and why?

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: $0.9 * 2 + 0.1 * 1 \approx 1.9$

ORACLE
BERKELEY DB



Question 1 - picking a storage engine

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 of $P=2^{26}$ entries. Assume $N=2^{40}$ and $B=2^{7}$. All internal nodes fit in memory. We are using a disk drive. What is your choice and why?

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

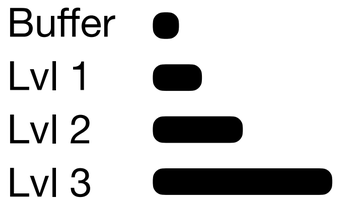
LSM-tree: A put costs $2 * (T/B) \cdot \log_T(N/P) \approx 0.66$ read & write I/Os

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

Avg. # I/Os per operation: $0.66 \cdot 0.9 + 4.2 \cdot 0.1 \approx 1.0$ I/Os (Cheaper)

Question 2 - tuning an LSM-tree

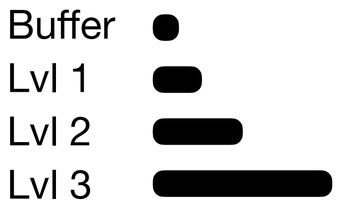
A friend tells you they switched from using a B-tree to a basic LSM-tree (with size ratio 2), yet write-amplification actually increased. There are $N=2^{40}$ entries, and the memtable size P and block size B are both $B=P=2^5$ entries. Explain why this happened. Identify three tuning options for reducing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



Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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: $(1/B) * \log_2(N/P) = 1.1$ write I/O (Costlier than B-tree)

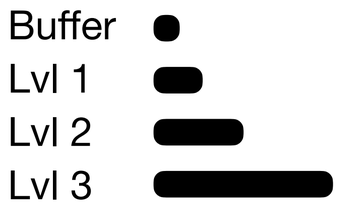


Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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: $(1/B) * \log_2(N/P) = 1.1$ write I/O (Costlier than B-tree)

Increasing the buffer size P reduces the number of levels and thus WA.

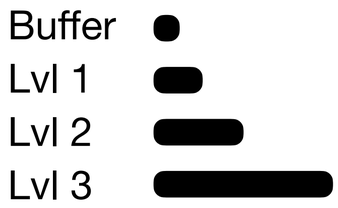


Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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: $(1/B) * \log_2(N/P) = 1.1$ write I/O (Costlier than B-tree)

Increasing the buffer size P reduces the number of levels and thus WA.



We can increase the buffer size to decrease the number of levels across which entries get merged. E.g., $\mathbf{P=2^{20}}$ entries leads to:

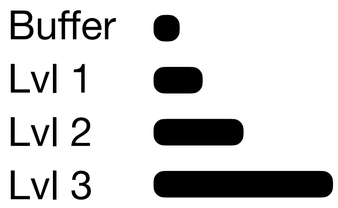
$$(1/B) * \log_2(N/\mathbf{P}) = 0.63$$

Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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: $(1/B) * \log_2(N/P) = 1.1$ write I/O (Costlier than B-tree)

Increasing the buffer size P reduces the number of levels and thus WA.



We can increase the buffer size to decrease the number of levels across which entries get merged. E.g., **$P=2^{20}$** entries leads to:

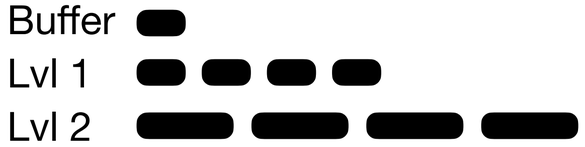
$(1/B) * \log_2(N/\mathbf{P}) = 0.63$ **Trade-off: requires more memory**

Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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**

e.g., $(1/B) * \log_{\mathbf{T}}(N/P) = (1/2^5) * \log_{\mathbf{10}}(2^{40}/2^{20}) = 0.19$

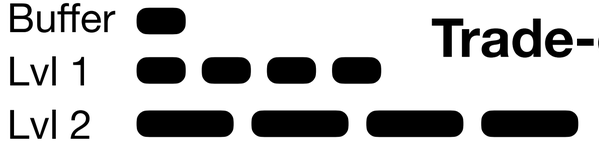


Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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**

e.g., $(1/B) * \log_{\mathbf{T}}(N/P) = (1/2^5) * \log_{\mathbf{10}}(2^{40}/2^{20}) = 0.19$



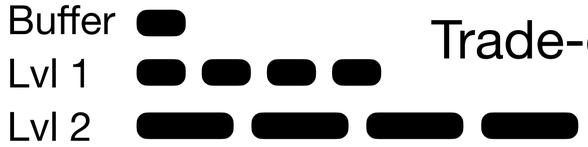
Trade-off: reads get more expensive.

Question 2 - tuning an LSM-tree

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 the memtable size P and block size B are both $B=P=2^5$ entries. 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

e.g., $(1/B) * \log_{\mathbf{T}}(N/P) = (1/2^5) * \log_{\mathbf{10}}(2^{40}/2^{20}) = 0.19$



Trade-off: reads get more expensive.

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

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	**Unclustered?**
#levels:	$L_C=\log_T(N/B)$	
put:	$O(L_C / B)$	
get:	$O(L_C)$	
scan:	$O(L_C+S/B)$	

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	**Unclustered?**
#levels:	$L_C=\log_T(N/B)$	**$L_U=\log_T(N/K)$**
put:	$O(L_C / B)$	
get:	$O(L_C)$	
scan:	$O(L_C+S/B)$	

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	**Unclustered?**
#levels:	$L_C=\log_T(N/B)$	$L_U=\log_T(N/K)$
put:	$O(L_C / B)$	**$O(1/B + L_U/K)$**
get:	$O(L_C)$	
scan:	$O(L_C+S/B)$	

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	**Unclustered?**
#levels:	$L_C=\log_T(N/B)$	$L_U=\log_T(N/K)$
put:	$O(L_C / B)$	$O(1/B + L_U/K)$
get:	$O(L_C)$	**$O(L_U + 1)$**
scan:	$O(L_C+S/B)$	

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	**Unclustered**
#levels:	$L_C=\log_T(N/B)$	$L_U=\log_T(N/K)$
put:	$O(L_C / B)$	$O(1/B + L_U/K)$
get:	$O(L_C)$	$O(L_U + 1)$
scan:	$O(L_C+S/B)$	**$O(L_U+S)$**

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	Unclustered	**Unclustered is better when…**
#levels:	$L_C=\log_T(N/B)$	$L_U=\log_T(N/K)$	
put:	$O(L_C / B)$	$O(1/B + L_U/K)$	
get:	$O(L_C)$	$O(L_U + 1)$	
scan:	$O(L_C+S/B)$	$O(L_U+S)$	

Question 3 - clustered vs. unclustered LSM-trees

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. Assume a basic LSM-tree (size ratio T=2), insertions only (no deletes or updates), and a 1 page memtable. 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 the two approaches shines.

	Clustered	Unclustered
#levels:	$L_C=\log_T(N/B)$	$L_U=\log_T(N/K)$
put:	$O(L_C / B)$	$O(1/B + L_U/K)$
get:	$O(L_C)$	$O(L_U + 1)$
scan:	$O(L_C+S/B)$	$O(L_U+S)$

Unclustered is better when…

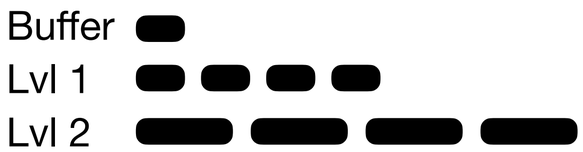
- Large data entries (i.e., $K > B$)
- Few long range queries (S is small)
- Gets are not a clear cut

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

e.g., T=4

Tiered



Leveled

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

e.g., T=4

Tiered

Leveled

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

$$\textbf{Space-amp} = \frac{\textbf{black + red}}{\textbf{black}}$$

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

$$\text{Space-amp} = \frac{\text{black} + \text{red}}{\text{black}} = \frac{\textbf{black} + \textbf{black} / \textbf{T} + \textbf{black} / \textbf{T}^2 + \ldots}{\textbf{black}}$$

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

$$\text{Space-amp} = \frac{\text{black + red}}{\text{black}} = \frac{\text{black + black / T + black / T}^2 + \ldots}{\text{black}} = 1 + \frac{1}{T} + \frac{1}{T^2} + \cdots \leq \frac{T}{T-1}$$

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

e.g., T=4

Tiered

???

Leveled

$$\leq \frac{T}{T-1}$$

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

Tiered

e.g., T=4

$$\text{Space-amp} = \frac{\text{black + red}}{\text{black}} = \frac{\text{(T-1) * black + black + black / T}^2}{\text{black}} \approx T$$

Question 4 - space-amplification

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 worst-case space-amplification for a leveled vs. tiered LSM-tree with size ratio T between any two adjacent levels. Assume all levels are totally full and all entries are equally sized.

e.g., T=4

Tiered

$\approx T$

Leveled

$$\leq \frac{T}{T-1}$$