SILK: Preventing Latency Spikes in Log-Structured Merge Key-Value Stores

O. Balmau*, F. Dinu*, W. Zwaenepoel*, K. Gupta†, R. Chandhiramoorthi†, D. Didona§

*THE UNIVERSITY OF SYDNEY  †NUTANIX™  §IBM Research | Zurich

USENIX ATC 2019
Log-Structured Merge (LSM) KVs

- Designed for write-heavy workloads
- Handle large-scale data
- Working set does not fit in RAM
Log-Structured Merge (LSM) KVs

- Designed for write-heavy workloads?
- Handle large-scale data
- Working set does not fit in RAM
LSM KV Latency Spikes in RocksDB

Nutanix write-intensive production workload

Lower is better

99p Latency (micros)

0 500 1000 1500 2000

Time (s)

10^3 10^6

oana.balmau@sydney.edu.au
Latency spikes of up to 1s in write dominated workloads.

Nutanix write-intensive production workload
Latency Spikes in LSM KVs

Why is this important?

❌ Cannot provide SLA guarantees to clients.

❌ Unpredictable performance when connecting LSM in larger pipelines.
Our Contribution: The SILK LSM KV

- Solves latency spike problem for write-heavy workloads.
- No negative side-effects for other workloads.
- SILK introduces the notion of an I/O scheduler for LSM KVs.
Experimental Study: Reason Behind Latency Spikes
What Causes LSM Latency Spikes?

Severe competition for I/O bandwidth between client operations and LSM internal operations (~GC).
LSM KV Overview
LSM KV Overview

SSTables
- sorted files
- many SSTables/Level

Write buffer

Write buffer

RAM

Disk
HDD/SSD
LSM KV Client Operations
Three types of internal ops:
1. Flushing
2. $L_0 \rightarrow L_1$ compaction
3. Higher level compactions

No coordination between internal operations.
LSM Internal Ops: Flushing

Flush when Write buffer full.
Incoming writes absorbed in new **write buffer**. **Flush buffer** written to L0.
LSM Internal Ops: \( L_0 \rightarrow L_1 \) compactions

Merge one \( L_0 \) SSTable with \( L_1 \).
LSM Internal Ops: L0 $\rightarrow$ L1 compactions

Merge one L0 SSTable with L1.
Makes room on L0 for flushing.
 LSM Internal Ops: Higher Level Compactions

~GC in the LSM tree.
Discard duplicates & delete values.
Less urgent than L0→L1 compactions.

... but need to complete.
I/O bandwidth intensive.

Memory

Disk

L_0

L_1

L_2

L_3

oana.balmau@sydney.edu.au
~GC in the LSM tree. Discard duplicates & delete values.

Less urgent than $L_0 \to L_1$ compactions.

... but need to complete.

I/O bandwidth intensive.

Can have many higher level compactions running in parallel.
LSM Review

Internal operations:

1. **Flushing.** From memory to disk.
2. **L0 → L1 compaction.** Make room to flush new files.
3. **Higher level compactions.** ~GC, I/O intensive.

![Alert]

*No coordination between internal ops and client ops.*
What Causes LSM Latency Spikes?

Both reads and writes experience latency spikes.

Focus on writes. Less intuitive.

Writes finish in memory. Why do we have 1s latencies?
L0 Full, Cannot Flush

update

Write buffer

Flush buffer

Memory

Disk

L0

L1

L2

L3

Update buffer

Write buffer

Flush buffer
L0 Full, Cannot Flush

No room to write on L0
L0 Full, Cannot Flush

No room to write on L0
L0 Full, Cannot Flush

Memory

Disk

update

Write buffer

Flush buffer

No room to write on L0

L1

L2

L3

oana.balmau@sydney.edu.au
L0 Full, Cannot Flush

Memory

Disk

Write buffer

Flush buffer

update

No room to write on L0

update

L_0

L_1

L_2

L_3
L0 Full, Cannot Flush

Memory | Disk
---|---
Update | Write buffer

No room to write on L0
1. Writes Blocked Because L0 is Full.

No coordination between internal ops.

Higher level compactions take over I/O.

L0 → L1 compaction is too slow.

Not enough space on L0.

Cannot flush memory component.
1. Writes Blocked Because L0 is Full.
1. Writes Blocked Because L0 is Full.

- **L0 is FULL**

- Higher level compaction
  - SLOW L0 → L1 compaction

- Time (seconds)
1. Writes Blocked Because L0 is Full.

**SLOW L0 → L1 compaction**

- **L0 is FULL**
  - Cannot flush. No space on L0

- Higher level compaction
- Higher level compaction
- Higher level compaction
1. Writes Blocked Because L0 is Full.

L0 is FULL

Higher level compaction

Higher level compaction

Higher level compaction

SLOW L0 $\rightarrow$ L1 compaction

Latency spike!

Time (seconds)
Flushing is Slow

Memory

Disk

update

Write buffer

Flush buffer

L₀

L₁

L₂

L₃

update

Write buffer

Flush buffer
Flushing is Slow

update

Memory

Write buffer

Disk

Flush buffer

L_0

L_1

L_2

L_3
Flushing is Slow

- Memory
- Disk

update

Write buffer

Flush buffer

L1

L2

L3

oana.balmau@sydney.edu.au
Flushing is Slow

Write buffer fills up before flush buffer is written to disk.
Flush buffer fills up before flush buffer is written to disk.
Flushing is Slow

Write buffer fills up before flush buffer is written to disk.

oana.balmau@sydney.edu.au
2. Writes Blocked Because Flushing is Slow.

No coordination between internal ops.

Higher level compactions take over I/O.

Flushing does not have enough I/O.

Flushing is very slow.

Memory component becomes full.
2. Writes Blocked Because Flushing is Slow.

Many parallel higher level compactions
2. Writes Blocked Because Flushing is Slow.

- Flush does not have enough I/O to finish fast
- Many parallel higher level compactions

Time (seconds)

1  2  3  4  5  6  7  8  9  10  11  12  13
2. Writes Blocked Because Flushing is Slow.

Time (seconds)

1 2 3 4 5 6 7 8 9 10 11 12 13

flush flush SLOW flush

Higher level compaction
Higher level compaction
Higher level compaction

Latency spike!

oana.balmau@sydney.edu.au
Naïve Solution 1: Compaction Rate Limiting

**Rate Limiting:** simple attempt to coordinate between internal and external ops.

![Graph showing latency in microseconds over time](image-url)

- **90 MB/s Max Compaction Bandwidth**
- **RocksDB**
Naïve Solution 1: Compaction Rate Limiting

Static compaction rate limiting does not work in the long term.
Chance to run many parallel high level compactions increases.
Naïve Solution 2: Delay Compaction Work

Selective/Delayed Compaction (TRIAD [USENIX ATC ’17], PebblesDB [SOSP ’17]).
Naïve Solution 2: Delay Compaction Work

Being selective about compactions does not avoid interference. Eventually need to do the delayed compaction work.
Lessons Learned

1. Make sure L0 is never full.
Lessons Learned

1. Make sure L0 is never full.

2. Ensure sufficient I/O for flush/compactions on low levels.
Lessons Learned

1. Make sure L0 is never full.

2. Ensure sufficient I/O for flush/compactions on low levels.

3. Higher level compactions should not fall behind too much.
The SILK I/O Scheduler
I/O scheduler for LSM KVs: coordinate I/O bandwidth sharing to minimize interference between internal ops and client ops.
Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/compactions on low levels.

Make sure other compactions do not fall behind too much.
Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Prioritize internal operations at lower levels of the tree.
Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Prioritize internal operations at lower levels of the tree.

Preempt higher level compactions if necessary.
Lessons Learned

Make sure L0 is never full.

Ensure sufficient I/O for flush/compactions on low levels.

Make sure other compactions do not fall behind too much.

SILK Design

Prioritize internal operations at lower levels of the tree.

Preempt higher level compactions if necessary.

Opportunistically allocate I/O for higher level compactions.
Prioritize & Preempt

Prioritize internal ops at lower tree levels:

1. Flushing
2. L0 → L1 compactions
3. Higher level compactions
Prioritize & Preempt

Prioritize internal ops at **lower tree levels:**

1. **Flushing** – *dedicated flush operation queue.*
2. L0 → L1 compactions
3. Higher level compactions
Prioritize & Preempt

Prioritize internal ops at **lower tree levels:**

1. **Flushing** – *dedicated flush operation queue.*

2. L0 $\rightarrow$ L1 compactions

3. Higher level compactions

$L0 \rightarrow L1$ compaction preempts higher level compactions.
Opportunistically allocate I/O for compactions

Real Nutanix client load example

![Graph](image)
Opportunistically allocate I/O for compactions

Real Nutanix client load example

Client workload is not constant.
Opportunistically allocate I/O for compactions

Real Nutanix client load example

Client workload is not constant.

SILK continuously monitors client I/O bandwidth use.
Opportunistically allocate I/O for compactions

Real Nutanix client load example

Allocate less I/O to compactions during client load peaks.
Allocate more I/O to compactions during low client load.
Opportunistically allocate I/O for compactions

Real Nutanix client load example

More I/O to high level compactions during low load → don’t fall behind.
SILK Evaluation
SILK Implementation

Extends RocksDB.

Open Source https://github.com/theoanab/SILK-USENIXATC2019
Benchmark with different workloads:
write-intensive, read-intensive, scan-intensive.

Show:
1. Write-heavy workloads: SILK is much better for tail latency.
2. Other workloads: SILK is not detrimental.
YCSB

Lower is better

99p Latency (micros)

- YCSB A: Write Dominated
- YCSB B: Read Dominated
- YCSB C: Read Only
- YCSB D: Read Latest
- YCSB E: Scan Dominated
- YCSB F: Update Dominated

- SILK 99p
- RocksDB 99p

Lower is better
SILK decreases tail latency by 4 orders of magnitude in write-dominated workloads.
YCSB

SILK does not affect read/scan dominated workloads

99p Latency (micros)

SILK 99p
RocksDB 99p

YCSB A
Write Dominated

YCSB B
Read Dominated

YCSB C
Read Only

YCSB D
Read Latest

YCSB E
Scan Dominated

YCSB F
Update Dominated

Lower is better

oana.balmau@sydney.edu.au
Write dominated:  
57% writes, 41% reads, 2% scans.

Bursty (open loop):  
Peaks and valleys in client load.

Dataset size:  500GB, KV tuple size 400B on average.
SILK vs RocksDB Tail Latency 99P

Lower is better
SILK vs RocksDB Tail Latency 99P

SILK achieves 3 orders-of-magnitude improvement in tail latency in production workloads.
SILK vs RocksDB Stalling

Lower is better

% time stalling

0 10 20

0s 100s

RocksDB SILK
SILK vs RocksDB Stalling

SILK never stalls because it can always do timely flushing.
More in the paper...

- More experiments and workloads.
- With SILK, throughput is steady and close to the client load.
- Comparison with more state-of-the-art LSMs (TRIAD, PebblesDB).
SILK Take-Home Message

• We introduce the **new concept** of an I/O scheduler for LSM.

• **Coordinate I/O sharing** to avoid latency spikes.

• **Three orders-of-magnitude improvements** on tail latency.

Thank you! Questions?