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

Talk MSR Seattle, July 2019
About me

PhD Student at the University of Sydney.

Advised by Willy Zwaenepoel.

Bachelor and Master in CS from EPFL.

Research interests:
storage systems, distributed systems, concurrent algorithms, parallelism.

Internships in Nutanix CA/Bangalore, HP Vertica, ABB Research.
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

99p Latency (micros)

Lower is better

Time (s)

0 500 1000 1500 2000

$10^3$ $10^6$
Latency spikes of up to 1s in write dominated workloads! Spikes are up to 3 orders of magnitude > median tail latency
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

Diagram:
- Write buffer
- RAM
- Disk HDD/SSD
- Levels: $L_0$, ..., $L_n$
LSM KV Client Operations

- **update**
- **read**
- **Wb**
- **L₀**
- **...**
- **Lₙ**

RAM

Disk HDD/SSD

-oana.balmau@sydney.edu.au
LSM Internal Ops

Three types of internal ops:
1. Flushing
2. L0 → L1 compaction
3. Higher level compactions

No coordination between internal operations.
LSM Internal Ops: **Flushing**

Flush when Write buffer full.
LSM Internal Ops: **Flush**ing

- **Update**
- **New write buffer**
- **Flush buffer**

Incoming writes absorbed in new write buffer.
Flush buffer written to L0.
LSM Internal Ops: $L_0 \rightarrow L_1$ compactions

Merge one L0 SSTable with L1.
LSM Internal Ops: L0 $\rightarrow$ L1 compactions

Memory
Disk

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 $L_0 \rightarrow L_1$ compactions.

... but need to complete.
I/O bandwidth intensive.
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.
- L3 level is high in hierarchy.
- 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.

⚠️ 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?**
Cannot Flush

Memory

Disk

update

Write buffer

Flush buffer

L_0

L_1

L_2

L_3
Cannot Flush

Update

Write buffer

Flushing buffer

Memory

Disk

No room to write on L0

L1

L2

L3
Cannot Flush

Memory

Disk

Write buffer

Flush buffer

No room to write on L0

update

L1

L2

L3
Cannot Flush

Memory

Disk

update

Write buffer

Flush buffer

No room to write on L0

L1

L2

L3
1. Writes Blocked Because L0 is Full.

No coordination between internal ops.

Higher level compactions take over I/O.

L0 $\rightarrow$ L1 compaction is too slow.

Not enough space on L0.

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

- [0, 1]: Flush
- [2, 3]: Higher level compaction
- [4, 6]: Higher level compaction
- [7, 9]: Higher level compaction
- [10, 15]: Slow L0 → L1 compaction

**Time (seconds):**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Cannot Flush

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

- Flush
- Higher level compaction
- Higher level compaction
- Higher level compaction
- SLOW L0 → L1 compaction

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

- SLOW L0 $\rightarrow$ L1 compaction
- Higher level compaction
- Higher level compaction
- Higher level compaction
1. Writes Blocked Because L0 is Full.

- **L0 is FULL**
- **Cannot flush. No space on L0**
- **Higher level compaction**
- **SLOW L0 → L1 compaction**
1. Writes Blocked Because L0 is Full.

Time (seconds)

0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16

flush  flush  flush

Higher level compaction
Higher level compaction
Higher level compaction

SLOW L0 → L1 compaction

L0 is FULL

Latency spike!
Flushing is Slow

Memory → Update → Write buffer

Disk

L₀

L₁

L₂

L₃

Flush buffer
Flushing is Slow

Memory

Write buffer

Disk

update

Flush buffer

L_1

L_2

L_3
Flushing is Slow

Memory

Disk

update

Write buffer

Flush buffer

L_1

L_2

L_3
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.
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.
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.
2. Writes Blocked Because Flushing is Slow.

Many parallel higher level compactions

Time (seconds)

flush

flush

Higher level compaction
Higher level compaction
Higher level compaction

Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction

Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction

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

Flush does not have enough I/O to finish fast

Many parallel higher level compactions

Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction

Time (seconds)
2. Writes Blocked Because Flushing is Slow.

Latency spike!

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
Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction
Higher level compaction
Naïve Solution 1: Compaction Rate Limiting

Rate Limiting: simple attempt to coordinate between internal and external ops.
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
SILK Key Idea

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.

SILK Design
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. **First priority:** Flushing

2. **Second priority:** L0 → L1 compactions

3. **Third priority:** 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 → L1 compactions**
3. **Higher level compactions**

\[ L0 \rightarrow L1 \text{ compaction preempts higher level compactions.} \]
2. Preempt

Dedicated flush queue:

Compaction queues:
2. Preempt

Dedicated flush queue:

Compaction queues:

Running:

Flush
2. Preempt

Dedicated flush queue:

Running:

Flush

Compaction queues:

L0 → L1

L2 → L3

L3 → L4

L2 → L3

L1 → L2

L2 → L3
2. Preempt

Dedicated flush queue:

Running:
- Flush

Compaction queues:
- L0 → L1
- L2 → L3
- L2 → L3
- L3 → L4
- L1 → L2
- L2 → L3

Preempt!
2. Preempt

Dedicated flush queue:

Compaction queues:

Running:

Flush

L0 → L1

L1 → L2

L2 → L3

L3 → L4
2. Preempt

Running:

😊 L0→L1 compactions never wait behind higher level compactions

Compaction queues:

- L2→L3
- L2→L3
- L2→L3
- L3→L4
- L0→L1
- L1→L2
- L2→L3

oana.balmau@sydney.edu.au
Opportunistically allocate I/O for compactions

Real Nutanix client load example

```
<table>
<thead>
<tr>
<th>Time (s)</th>
<th>I/O Bandwidth (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
```

Client I/O
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 peaks**.

Allocate **more I/O to compactions** during **client low 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.
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.

Even in peak load, guarantee min I/O for flushing and L0 → L1 compaction.
SILK Evaluation
SILK Implementation

Extends RocksDB.

Open Source https://github.com/theoanab/SILK-USENIXATC2019
YCSB

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 Benchmark

SILK 99p

RocksDB 99p

SILK decreases tail latency by 4 orders of magnitude in write-dominated workloads.

Lower is better

Latency (micros)

10^6

10^4

10^2

YCSB A
Write Dominated

YCSB B
Read Dominated

YCSB C
Read Only

YCSB D
Read Latest

YCSB E
Scan Dominated

YCSB F
Update Dominated

oana.balmau@sydney.edu.au
76
YCSB Benchmark

SILK 99p

RocksDB 99p

SILK does not affect read/scan dominated workloads

Latency (micros)

Lower is better

YCSB A
Write Dominated

YCSB B
Read Dominated

YCSB C
Read Only

YCSB D
Read Latest

YCSB E
Scan Dominated

YCSB F
Update Dominated
YCSB Benchmark Median Latency

Lower is better

<table>
<thead>
<tr>
<th></th>
<th>SILK 50p</th>
<th>RocksDB 50p</th>
</tr>
</thead>
<tbody>
<tr>
<td>YCSB A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YCSB B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YCSB C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YCSB D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YCSB E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YCSB F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominated</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
YCSB Benchmark Median Latency

Latency (micros)

- SILK 50p
- RocksDB 50p

SILK does not affect median latency.

Lower is better

YCSB A
Write Dominated

YCSB B
Read Dominated

YCSB C
Read Only

YCSB D
Read Latest

YCSB E
Scan Dominated

YCSB F
Update Dominated

oana.balmau@sydney.edu.au
Nutanix Production Workload

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

- RocksDB
- SILK

Lower is better

99p Latency (micros)

Time (s)

0 500 1000 1500 2000
SILK for Nutanix Production Workload 24h

99p Latency
(micros)

Time (h)

$10^3$

$10^6$
Breakdown of SILK Techniques

- Dynamic I/O Rate Limiting
- Scheduling and Preemption
- SILK
SILK vs RocksDB Stalling

% time stalling

100s

0s

RocksDB

SILK

Lower is better
SILK for Nutanix Production Workload 24h
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.
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?