Hi!

I’m Remi (Remington Brasga).

As a Sr. Software Engineer for the Memory Storage Strategy Division at Toshiba Memory America, Inc., I am dedicated to development and collaboration on open source software. I earned my B.S. and M.S. from University of California, Irvine.

Today, I am here fresh from having my 3rd son who has kept me very busy the past few weeks.

(thanks for letting me come here and rest!)
Applications are King

• Software is eating the world

• Storage is “Defined by Software”
  - i.e., applications define how storage is used

• Yet, applications do not always use storage wisely
For Example, RocksDB

RocksDB is a popular data storage engine

...used by a wide range of database applications
The Challenge of Rocks

While highly effective as a data storage engine, RocksDB has some challenges for SSDs:

Write Amplification is very large 20-30X (or more) as a result of the compaction layers rewriting the same data.

This can result in impact to SSD endurance.
How does RocksDB work?

Keys and Values are stored together in pairs.

As more data is added, these pairs waterfall through the levels.

This waterfall occurs during each compaction.
Let’s Demonstrate
RocksDB…the Old Way…Log Structured Merge

- CPU
- Key & Value
- Key & Value
- Key & Value

- DRAM
- New key value database entries

- LSM
- Compaction Level 1
- Compaction Level 2
- Compaction Level 3

Saved to SSD in compaction level 1
RocksDB...the Old Way...Log Structured Merge

- More writes to compaction level 1
- Compaction level 1 fills up

- Key & Value
- Key & Value
- Key & Value
- Key & Value
- Key & Value
- Key & Value
- Key & Value
- Key & Value

CPU

DRAM

LSM

Compaction Level 1

Compaction Level 2

Compaction Level 3
RocksDB...the Old Way...Log Structured Merge

Compaction level 1 is full; Written to RAM...

Consolidated for compaction level 2
RocksDB...the Old Way...Log Structured Merge

- Compaction Level 2
- Compaction Level 3
- Compaction Level 1

Key & Value

Consolidated data written to compaction level 2

Application-generated write amplification

SSD ABUSE!
RocksDB...the Old Way...Log Structured Merge

Compaction level 2 filling up

Application-generated write amplification

Compaction Level 1

Compaction Level 2

Compaction Level 3

Key & Value

Key & Value

Key & Value

Key & Value

Key & Value

Key & Value

Key & Value

Key & Value
RocksDB...the Old Way...Log Structured Merge

CPU

DRAM

Compaction Level 2 fills up

Written to DRAM, and consolidated

Compaction Level 2

Compaction Level 3
RocksDB…the Old Way…Log Structured Merge

Written to compaction level 3

Application-generated write amplification

CPU

DRAM

Compaction Level 2

Compaction Level 3
As you can see,

- This results in rewriting data “that doesn’t change” during compaction
- This can have a significant impact on SSD health and endurance

The good news is...

There is a better way

Toshiba Memory America has re-architected RocksDB to solve this SSD endurance challenge
Inspiration

We looked at some of the academic work on Rocks

- Major inspiration came from the “wisckey” from the university of Wisconsin
  WiscKey: Separating Keys from Values in SSD-conscious storage

Coding Work

Significant code was involved to split the keys from the values

- Replaced the LSM key value pairs with the ring buffer (for a Value Log - VLog)
- The VLog is a large number of files,
  - with each file having about the same number of values as an SST* has keys
- VLogFiles are organized into FIFO structures called VLogRings

- One tricky issue - keeping track of the state of the VLog over a restart
- Active recycling added to compactions to reduce instances of uncompacted files

*SST=Stored String Table
Coding Work

Work involved managing the keys

• While keys mostly are cached in memory
• They still need to be written into LSM layers

• Compactions still occur,
  – about one per second as usual but only on the keys

• Coding for compaction involved
  – Compacting the keys
  – And cleaning up the stale data in the ring buffers
Performance Tuning

Tuning performance in the code was a challenge

- Some changes made performance worse (against every expectation...)

- The split keys and values approach needed new test cases

- There are a number of new option settings to enable optimizations
  this added improvements to
  - Compaction picker improved (to reduce need of active recycling)
  - Write amp and space amp measuring
  - Bulk load efficiency
RocksDB continues to evolve

- TRocksDB is maintaining version compatibility with each current revision of RocksDB
- TRocks is fully compatible and cross compiles with/into regular RocksDB code today
- Once compiled Rocks or TRocks can be enabled with option settings
How does TRocksDB work?

Keys and Values are split

Keys go into compaction layers (in memory)

Values are stored separately into a ring buffer
The Improved TRockDB Method
TRocksDB…the New Way Ring Buffer, Separate Keys/Values

CPU

DRAM

Key Value Key Value Key Value Key Value Key Value

Create new database entries

LSM

Ring Buffer
TRocksDB...the New Way Ring Buffer, Separate Keys/Values

CPU
Key Value Key Value Key Value Key Value

DRAM
Key Value LSM

Ring Buffer

Write Values to Ring Buffer

Keys stored separately (cacheable)

SSD

SSD endurance improved via lowered Write Amp

Lookups are lightening fast with keys cached in DRAM
Our Goal during Development

Improved (reduced) Write Amplification with the same or better performance
**Improved Write Amplification**

---

**Write Amplification**

*(lower is better)*

- **Test 1:** Random Bulk Load
  - TRocks 1.2.5.23: 0.6 (-40%)
  - Rocks FB 6.4: 1
- **Test 2:** Bulk Sequential Load
  - TRocks 1.2.5.23: 0.6 (-40%)
  - Rocks FB 6.4: 1
- **Test 3:** Random Overwrites
  - TRocks 1.2.5.23: -57%
  - Rocks FB 6.4: 4.9

---

Performance results based on internal Toshiba Memory testing. Benchmark tests were developed by Facebook and generated in July 2018. They measure RocksDB performance when data resides on flash storage and tested using the newest release RocksDB 6.1.fb. The benchmark information and test descriptions are available at [https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks](https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks).
Performance Comparison
(lower is better)

Test 1: Random Bulk Load
-28% 21,565

Test 2: Bulk Sequential Load
-11% 6,155

Test 3: Random Overwrites
-3% 9,595

Test 4: Random Key Reads
-24% 118,244

Performance results based on internal Toshiba Memory testing. Benchmark tests were developed by Facebook and generated in July 2018. They measure RocksDB performance when data resides on flash storage and tested using the newest release RocksDB 6.1.fb. The benchmark information and test descriptions are available at https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks.
Tuning and Optimization

• Every database is different

• Rocks and TRocks have a wide range settings / options to optimize the database

• These settings are often confusing or “left in default”
Tuning and Optimization

- Toshiba Memory America has created a “tuning” tool
  - It is on a Google spreadsheet
    - Available from the GitHub®
  - Easy to use - just fill in the blanks
  - It generates known good settings
Call to Action

Download

Test

Use

Enjoy!

Join the project
improve and contribute to the code
Developed by Toshiba Memory America
(Which will be renamed KIOXIA America, Inc. as of October 1, 2019)

Open Source

Improved write amplification = SSD last longer with the same or better performance

Available today:
https://github.com/ToshibaMemoryAmerica

All company names, product names and service names may be trademarks of their respective companies.