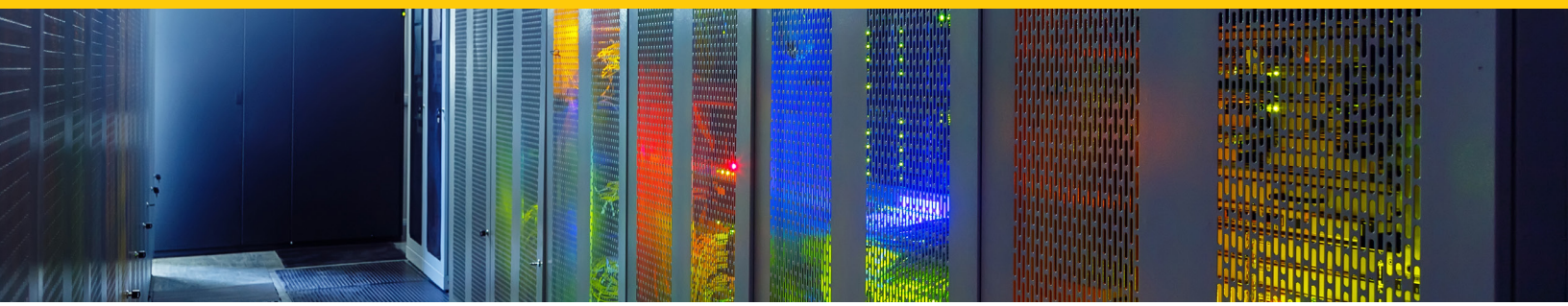
KIOXIA



Introducing the TRocksDB Platform

Enabling Improvements in Write Amplification and the Reduction of Repeated Data Rewriting to Optimize SSD Use with RocksDB

Scott Stetzer Tyler Nelson Scott Harlin

Editor's Note:
KIOXIA America, Inc. has updated this Tech Brief that was originally published and distributed at the Storage Developer's Conference in September 2019. It has been updated to provide current and relevant test data as our intention is to use the latest revision of RocksDB to compare to our TRocksDB code. To this end, we used an alpha version of RocksDB 6.4.fb development code. Given that the code we used for testing purposes was not 'released code,' KIOXIA America has taken the necessary steps to revise the data and use the same version of RocksDB (v6.1.2) that the TRocksDB code is based upon. In order to show the test data as transparently as possible, the test results include both RocksDB 6.1.2 data and RocksDB 6.4.fb alpha data (from the previously published Tech Brief).

Introduction

RocksDB is a high-performance embedded database for key-value data that can be used as a storage engine within a larger database management system (DBMS). It was developed by Facebook™ in April 2012, where they built it from the original LevelDB open-sourced, on-disk key-value store platform initially written by Google®. RocksDB was designed to accomplish tasks above and beyond LevelDB as follows:

- Improve server workload performance
- Run on servers with many CPU cores
- Use fast storage efficiently (such as flash-based SSDs)
- Support I/O-bound, in-memory, write-once workloads

The database has the ability to deliver massive scalability because RocksDB does not require database locks to invoke a read or write operation. This enables users to generate as many threads as they need to access the database without lock contention. Storage capacity is conserved through a layered data compaction scheme, where layers are of a defined size that enable the efficient and immediate access to hot data. This allows the database to be adaptable and highly available to different types of workloads.

It is of no surprise that the RocksDB platform has enjoyed success as a LevelDB replacement. Yet, there are still limitations and other challenges that require another step forward. Introducing TRocksDB - a new database design that enables improvements in storage and DRAM usage while reducing repeated data rewriting caused by application-generated write amplification (WA). TRocksDB is extremely flexible and designed for use with any off-the-shelf SSD.

Evolving RocksDB to a Higher Level

The RocksDB design utilizes a Log-Structured Merge (LSM)-tree to store data. The LSM-tree is a data structure with performance characteristics that make it attractive for providing indexed access to files with high insertion volume, such as log data. It rewrites data at least one time for every level of the database, and in many cases, multiple times per level. As a result, the total write amplification for RocksDB will be at least 21x for a single key-value pair. The WA generated causes application-level performance delays and early SSD wear-out.

Enterprise SSDs are designed with limits as to how much data can be written to each and regarded as Drive Writes per Day (DWPD). In the case of RocksDB, its LSM-tree structure design is responsible for the 21x write amplification, and when run in a steady state for longer intervals, will increase the WA even more, requiring higher levels of repeated data rewriting that will negatively impact DWPD.

In other words, if one terabyte (1TB)[1] of data is inserted into the RocksDB platform daily, an SSD could experience up to 21TB of data rewrites (1TB multiplied by 21x). If write amplification can be reduced, then the overall write workload to each SSD will be reduced, enabling the SSD to record new data and still remain within its recommended DWPD.

As RocksDB generates many WA-impacting rewrites, the workaround to maintain high-performance and still enable new data to be recorded has typically been to purchase more SSDs, and more servers in which to put them. More switches to connect servers to storage devices, and more admins and technicians to keep everything running smoothly could also be required, paving the way for new solutions like TRocksDB.

Enter TRocksDB

TRocksDB is being developed as an open-source improvement to RocksDB utilizing fast SSD storage, with support for I/O-bound, in-memory, write-once workloads. The TRocksDB design intent is to improve storage usage and performance by reducing application-generated WA and SSD wear, all while improving DRAM use for better key caching. Its open-source approach comes with a version-controlled wiki that enables development code to be readily available and downloadable from a GitHub® platform[2].

Though designed with similar performance expectations when compared to the RocksDB platform, TRocksDB features key architectural advancements. The most significant advancement enables values to be stored in separately-managed files (Value Logs or VLogs), where only the keys, not the values, are stored in the LSM-tree. Separating keys from values produces faster and more efficient database lookups. In the TRocksDB design, when values are much larger than keys, a low write amplification can be delivered by utilizing DRAM as a database cache.

To take this one step further, TRocksDB uses an LSM-tree as a base and identifies the key through a lookup table. A reference pointer is used to find the value of the key in a ring buffer. By caching the keys in the LSM-tree, it enables the very small values and reference pointers to be kept in DRAM. Since all of the bloom filters are part of the LSM-tree, the keys can be identified much faster. In RocksDB, the keys and values are stored together with no mechanisms in place to simplify the caching of keys. On the other hand, TRocksDB separates values and keys.

Through predictive models, key caching works best in the TRocksDB platform when the value size is at least two times larger than the key size. In most cases, the value is at least 10x larger, and is more likely to be 20x to 500x larger in a typical database use case. The larger the value, the more benefits the TRocksDB platform delivers over traditional key-value stores like RocksDB. These benefits include the ability to cache more levels of the database, decrease reads from storage, and reduce application-generated write amplification, all while delivering comparable or better SSD performance.

Benchmark Test Criteria

To validate the TRocksDB benefits over RocksDB, benchmark tests were conducted by KIOXIA Corporation (formerly Toshiba Memory Corporation) in a lab environment, comparing application-generated write amplification and system performance. The current version of TRocksDB under development (v1.2.5) - a current release of RocksDB (v6.1.2) - and an alpha version of the latest development code (v6.4.fb), were all used for testing purposes.

The main purpose of the tests were to determine the number of repeated data rewrites recorded within RocksDB (using an LSM-tree structure) versus TRocksDB and its ring buffer design. A reduction in application-generated WA will also reduce the write-specific workload so that an SSD can record new data while still remaining within its recommended DWPD. A secondary purpose of the tests verify further performance advantages over RocksDB that could enable even more system capabilities.

The internal test environment developed by KIOXIA Corporation provided a common baseline to compare TRocksDB performance versus RocksDB on the host-side and utilized separate use case benchmarks that were developed by Facebook for their internal testing purposes. The performance-specific benchmark tests used for comparison included:

- Test #1: Bulk load of 8 billion keys in random order
- Test #2: Bulk load of 8 billion keys in sequential order
- Test #3: Random write of 8 billion keys
- Test #4: Random read of 8 billion keys

For each performance test, an additional write amplification test was conducted based on the performance results. Since Test #4 is a read test, there is not a WA to report. A description of each test[3] now follows:

Test #1: Bulk load of 8 billion keys in random order

This test loads a large dataset of 8 billion keys into each database. The keys are inserted in **random order** since the database is empty at the beginning of the test run. There is no data being read when data loading begins. As each database is gradually loaded, both TRocksDB and RocksDB were configured to first load all of the data in Level-0 using an unsorted vector memtable[4] with compaction layers switched off. Each database then made a second pass over the data to merge and sort all of the files in Level-0, into sorted files in Level-1. The tests were configured for Snappy compression[5] and results included the time it took to load 8 billion random keys and compact the entire database. Since load time was measured, a lower time-to-complete value equates to faster performance.

Once this performance test was completed, the WA test was conducted on each database using internally- generated reporting statistics available within the RocksDB platform and the Test #1 performance results.

Test #2: Bulk load of 8 billion keys in sequential order

This test is identical to Test #1 above except that the 8 billion keys were inserted in **sequential order**. As the database is gradually filled, both TRocksDB and RocksDB were configured to use multi-threaded compactions so that multiple threads could simultaneously compact non-overlapping key ranges in multiple levels via file renames - a primary reason why RocksDB is much faster than LevelDB for this kind of workload. The results for this test included the time it took to load 8 billion keys into each database sequentially with multi-threaded compactions occurring inline. Since load time was measured, a lower value equates to faster performance.

Once this performance test was completed, the WA test was conducted on each database using internally- generated reporting statistics available within the RocksDB platform and the Test #2 performance results.

Test #3: Random write of 8 billion keys:

This test randomly overwrites 8 billion keys into each database and measures performance. Using the set-up configuration from Test #2, each database loaded 8 billion keys sequentially and used a single thread with Write-Ahead-Log (WAL) enabled when the tests were run. Both databases were configured with 20 compaction threads where the threads could simultaneously compact non-overlapping key ranges in the same or in different levels.

Both databases were configured to support 1TB of storage capacity by setting the number of levels to 6 so that write amplification could be reduced. The Level-0 and Level-1 compactions were given priority to reduce stalls. Additionally, Snappy compression was only enabled for levels 2 and higher so that Level-0 compactions could occur faster. Files were configured in 64MB[1] sizes. The results included the time it took to load a database with 8 billion keys while overwriting 2 billion keys in random order. As Test #3 is also a load test, a lower time-to-complete value equates to faster performance.

Once this performance test was completed, the WA test was conducted on each database using internally-generated reporting statistics and the Test #3 performance results.

Test #4: Random read of 8 billion keys:

This test measures **random read** performance of each database utilizing 1 billion keys where each key is 20 bytes and each value is 400 bytes. Both databases were configured with a 4KB block size and with data compression enabled using Snappy compression. These tests utilized single-threaded compactions that issued approximately 1 billion random key reads to each database, and were configured to verify checksums on every read operation.

Data was first loaded into each database by sequentially writing all 8 billion keys. Once the load was completed, the benchmark test randomly picked a key and issued a read request. The measurement for this test **DOES NOT** include data loading - only the part that issued the random read request to each database was measured. The tests results included the time it took to randomly read 1 billion keys from an 8 billion key database. As Test #4 is a read performance test, a shorter completion time is better, and there is no WA to report.

4a Test Equipment:
The hardware and software equipment used for the benchmark tests included the following:

- **Databases Tested:**
  TRocksDB version 1.2.5.
  RocksDB version 6.1.2 and RocksDB version 6.4.fb (alpha).
- **Server:** Two (2) Tyan AMD® EPYC 7401P single socket servers featuring twenty-four (24) processing cores, 2.20 GHz frequency, and 128GB of DDR4 RAM.
- **Operating System:** Ubuntu® 18.04.
- **Application:** Database (particularly loading and overwriting of keys).
- **Storage Devices:** Six (6) NVMe™ SSDs in Linux® RAID0 with 450GB capacities and running an XFS file system.
- **Benchmark Test Software:** DB_Bench (a benchmark tool built into each database platform). Since RocksDB is a storage engine, traditional database tests do not work very well on this platform. As such, DB_Bench runs directly against the RocksDB storage engine for testing purposes.

4b Test Procedures:
The benchmark tests measured both TRocksDB and RocksDB performance based on standard Facebook tests and utilized previously developed Facebook commands to load data into each database. The results of each test were obtained from the DB_Bench benchmark tool that ran the tests directly against the TRocksDB and RocksDB storage engines. Once each performance test was completed, the average result from three test runs were used for comparison. From this result, the WA test was conducted on each database using their internal logging mechanism that captured the WA every minute, and then output them to log files.

Test Results

The test results[6] are divided into the four benchmark tests that were conducted with the average test run values represented in Chart 1 below. A breakdown of each test result is also included.

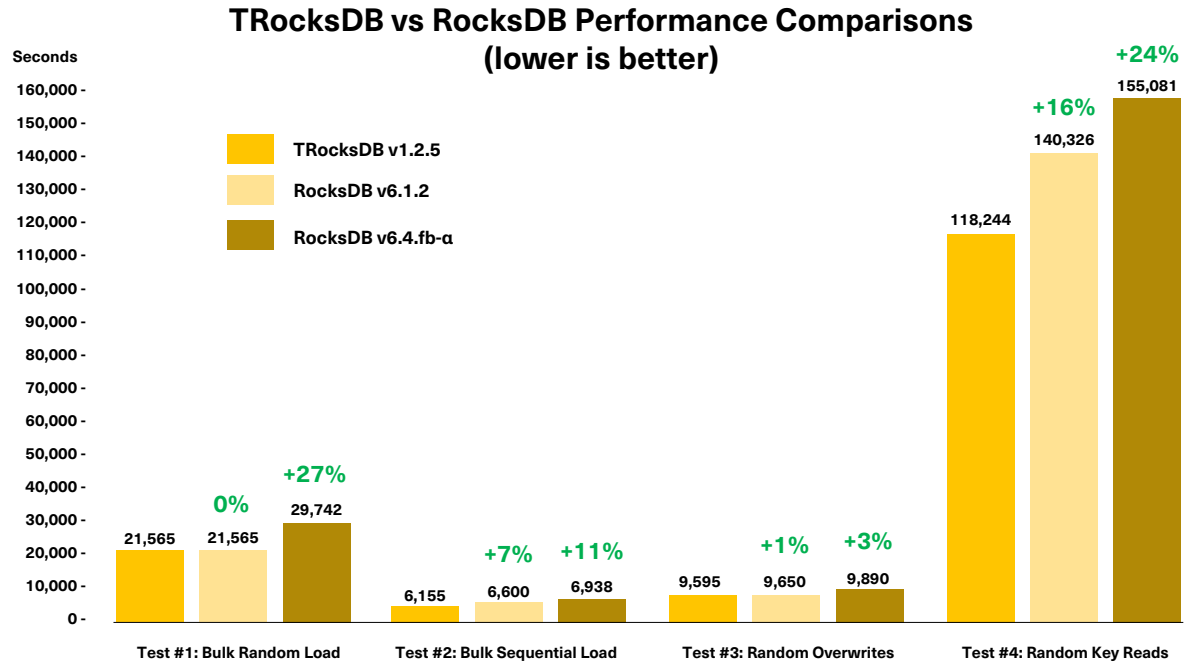


Chart 1: Performance comparisons in four key areas of database loading and overwriting
(percentage differences depicted in green)

From each performance result in Chart 1 above, a write amplification test was conducted from internally-developed software that captured the WA every minute. The WA results from each test are depicted in Chart 2.

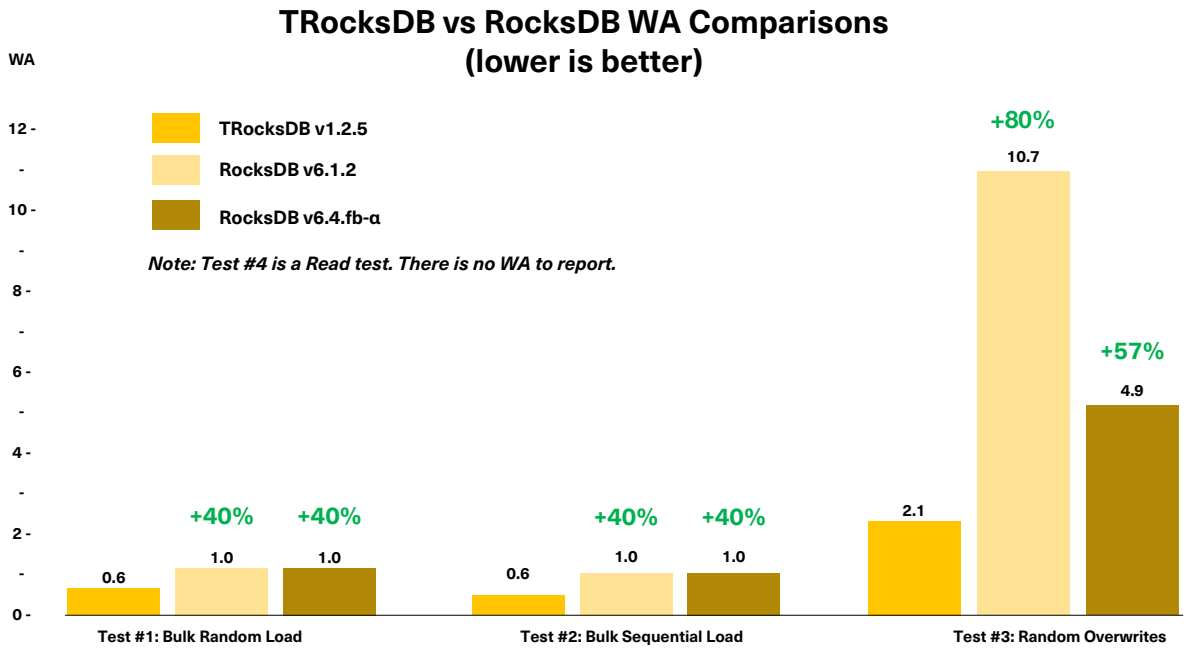


Chart 2: Write amplification comparisons in three key areas of performance
(percentage differences depicted in green)

5a Test #1 - Bulk load of 8 billion keys in random order

This test measured the time it took to load a large dataset of 8 billion keys into each database in random order as outlined in Section 4 Benchmark Test Criteria. Once this performance test was completed, the WA test was conducted on each database using the test results.

Database Platform	Best Result (in seconds)	TRocksDB Advantage	WA Result	TRocksDB Advantage
TRocksDB v1.2.5	21,565		0.6	
RocksDB v6.1.2	21,565	0%	1.0	~+40%
RocksDB v6.4.fb-α	29,742	~+27%	1.0	~+40%

5b Test #2 - Bulk load of 8 billion keys in sequential order

This test measured the time it took to load a large dataset of 8 billion keys into each database in sequential order as outlined in Section 4 Benchmark Test Criteria. Once this performance test was completed, the WA test was conducted on each database using the test results.

Database Platform	Best Result (in seconds)	TRocksDB Advantage	WA Result	TRocksDB Advantage
TRocksDB v1.2.5	6,155		0.6	
RocksDB v6.1.2	6,600	~+7%	1.0	~+40%
RocksDB v6.4.fb-α	6,938	~+11%	1.0	~+40%

5c Test #3 – Random write of 8 billion keys

This test measured the time it took to load a large dataset of 8 billion keys into each database in sequential order while overwriting 2 billion keys in random order, as outlined in Section 4 Benchmark Test Criteria. Once this performance test was completed, the WA test was conducted on each database using the test results.

Database Platform	Best Result (in seconds)	TRocksDB Advantage	WA Result	TRocksDB Advantage
TRocksDB v1.2.5	9,595		2.1	
RocksDB v6.1.2	9,650	~+1%	10.7	~+80%
RocksDB v6.4.fb-α	9.890	~+3%	4.9	~+57%

5d Test #4 – Random read of 8 billion keys

This test measured the time it took to randomly read 1 billion keys from an 8 billion key database as outlined in Section 4 Benchmark Test Criteria. Since this test strictly reads data, there is no WA to report and no write amplification test was conducted.

Database Platform	Best Result (in seconds)	TRocksDB Advantage	WA Result	TRocksDB Advantage
TRocksDB v1.2.5	118,244		N/A	
RocksDB v6.1.2	140,326	~+16%	N/A	N/A
RocksDB v6.4.fb-α	155,081	~+24%	N/A	N/A

Assessment

From the results of the benchmark tests, the performance of the TRocksDB platform was comparable or exceeded that of the current released version of RocksDB (v6.1.2). Furthermore, a reduction in repeated data rewrites was achieved. These performance and WA advantages include:

- Comparable performance with up to a 40% WA improvement when randomly loading 8 billion keys
- Up to 7% higher performance with up to a 40% WA improvement when sequentially loading 8 billion keys
- Comparable performance with up to an 80% WA improvement when randomly writing 8 billion keys
- Up to 16% higher performance when randomly reading 8 billion keys

Test Analysis

Based on predictive models that compared a current version of RocksDB to an early-developed draft of TRocksDB, it was determined that RocksDB rewrites every byte recorded by the database and writes or rewrites data to the SSD about 21x due to the LSM-tree and compaction layer architecture. Simply put, for every byte recorded by the database, 21 bytes are written to the SSD.

When TRocksDB was run under the same predictive model, its application-generated WA was about 3x, or for every byte recorded by the database, only 3 bytes are written to the SSD. Strictly based on predictive models, the TRocksDB platform, even in an early development stage, reduced application-generated WA from about 21 write operations down to 3.

The performance and write amplification tests were conducted to validate the predictive models using real world use cases. From a performance perspective, tests were conducted that loaded a database with billions of keys or overwrote data or randomly selected data. Based on the majority of results in the five tests, TRocksDB verified performance advantages over RocksDB.

The write amplification tests determined the number of repeated data rewrites that were recorded within RocksDB (using its LSM-tree structure design) versus TRocksDB and its ring buffer design. Based on the majority of results in the five tests, TRocksDB reduced the application-generated WA significantly, which in turn, reduced write-specific workloads so that SSDs deployed within the TRocksDB platform will be able to record new data and still remain within their recommended DWPD.

Summary

If an application causes every write operation to be re-written many times within a database, the amount of new data that can be written to an SSD can be significantly reduced. Those that are addressing this challenge today, whether they be IT professionals, database administrators, system architects, data analysts and scientists, or chief data officers, will be interested in cost-effective solutions. At present, a typical workaround would be to purchase more write-intensive SSDs, more servers to put them in, more switches to connect them, and more admins and techs to keep them running. All of which increase total operating cost (TOC).

Though RocksDB has been a successful replacement to LevelDB, it rewrites data at least one time for every level of the database, and in many cases, multiple times per level. Given a single key-value pair, the total write amplification for RocksDB will be at least 21x resulting in application-level performance delays and early SSD wear-out.

The new TRocksDB solution (under development at KIOXIA Corporation) is a database platform that includes server-side software and fast-performing SSDs designed to reduce repeated data rewriting caused by application-generated write amplification. It has the unique ability of separating keys from values, which in turn, enables very fast and efficient database lookups. Based on the internal benchmark tests conducted, TRocksDB delivered a lower write amplification for the system as a whole and comparable or better random and sequential performance when tested against RocksDB.

The TRocksDB platform responds to database queries faster and more efficiently then RocksDB as it uses DRAM as a database cache. In RocksDB, the keys and values are stored together with no mechanisms in place to simply cache keys. TRocksDB separates values and keys, and from the internal benchmark tests, reduced the associated WA by up to 80% versus RocksDB.

The TRocksDB platform includes server software that is now available under the terms of open-sourced licensing. The software is intended to work with fast-performing enterprise SSDs from any vendor and will run on any Linux hardware that RocksDB runs today.

The TRocksDB release candidate is now available on GitHub at: https://github.com/KioxiaAmerica/trocksdb.

Disclaimer: KIOXIA America, Inc. (KAI) may make changes to specifications and product descriptions at any time. The information presented in this technical brief is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors. Any performance tests and ratings are measured using systems that reflect the approximate performance of KAI products as measured by those tests. Any differences in software or hardware configuration may affect actual performance, and KAI does not control the design or implementation of third party benchmarks or websites referenced in this document. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to any changes in product and/or roadmap, component and hardware revision changes, new model and/or product releases, software changes, firmware changes, or the like. KAI assumes no obligation to update or otherwise correct or revise this information.

KAI makes no representations or warranties with respect to the contents herein and assumes no responsibility for any inaccuracies, errors or omissions that may appear in this information.

KAI specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. In no event will KAI be liable to any person for any direct, indirect, special or other consequential damages arising from the use of any information contained herein, even if KAI is expressly advised of the possibility of such damages.

Notes:

[1] Definition of capacity: KIOXIA Corporation defines a megabyte (MB) as 1,000,000 bytes, a gigabyte (GB) as 1,000,000,000 bytes and a terabyte (TB) as 1,000,000,000,000 bytes. A computer operating system, however, reports storage capacity using powers of 2 for the definition of 1GB = 2^{30} bytes = 1,073,741,824 bytes, 1TB = 2^{40} bytes = 1,099,511,627,776 bytes and therefore shows less storage capacity. Available storage capacity (including examples of various media files) will vary based on file size, formatting, settings, software and operating system, and/or pre-installed software applications, or media content. Actual formatted capacity may vary.

[2] The GitHub branch platform includes a website and cloud-based service that stores and manages the code that a company develops, and tracks and controls any change to their code.

[3] These performance benchmarks were developed by Facebook and generated in July 2018. They measure RocksDB performance when data resides on flash storage and tested using RocksDB 6.1.2 and an alpha version of RocksDB 6.4.fb. The benchmarks and test descriptions are available at https://github.com/facebook/rocksdb/wiki/Performance-Benchmarks.

[4] A memtable is an in-memory write-back cache of data rows that stores content as a key/column and can be looked up by a key.

[5] Snappy is a fast data compression and decompression library written in C++ by Google and does not aim for maximum compression, or compatibility with any other compression library, but instead, aims for high speeds and reasonable compression.

[6] The test results were captured and accurate at the time of the testing and may vary with time as software and hardware technology evolves.