White Paper

Optimizing TRocksDB for Improved Performance

Recommended Tuning Options for CPU Threads, Background Jobs, Write Buffers, File and Cache Sizes, and more

Tyler Nelson Scott Harlin

Introduction

TRocksDB, developed by KIOXIA America (formerly Toshiba Memory America), is an enhanced version of the popular RocksDB database developed by Facebook™. In TRocksDB, the database file is separated into two sets of files - one for keys and one for values. The benefit, with SSD storage in mind, is to reduce application induced write amplification (WA) to improve the life and health of SSDs[1].

TRocksDB is different in the way it uses separate key and value files, and thus, compaction efficiency is improved versus the core RocksDB platform. To achieve optimized performance and increased value from TRocksDB, a variety of its parameters within the configuration files can be adjusted. The key principles and elements required for tuning are also presented, showcasing the tuning configurations and recommended settings for achieving optimal TRocksDB performance. These configurations and settings are validated by thousands of hours of internal testing. In addition, an online spreadsheet tool referred to as the Optimized Tuning Guide, is introduced to assist with the tuning process.

Design Recap: RocksDB vs TRocksDB

RocksDB utilizes a log-structured merge (LSM)-tree to deliver high-performance storage. The LSM tree structures data with performance characteristics that make it attractive for providing indexed access to files with high insertion volume, such as transactional log data. Similar to other search trees, LSM trees maintain key-value[2] pairs that are stored together with no mechanism to cache the keys separately. As a result, data has to be rewritten at least once for each LSM level of the database. Given just one key-value pair, the total WA for RocksDB could be 21x or more for certain workloads. That not only adversely effects application-level performance, but may also result in early wear-out of the storage media itself.

In TRocksDB, keys and values are separated and no longer share the same file. Values are placed in the Value Log (VLog) while keys still traverse through the traditional RocksDB LSM-tree structure. The ability to separate keys from values enable very fast and efficient database lookups, as well as faster and more efficient database queries, especially when compared to RocksDB[3] which uses DRAM as a database cache. It also dramatically minimizes the WA required.

TRocksDB Tuning Principles

To tune TRocksDB for optimal performance, the file sizes it uses need to be decreased from the file sizes normally used in RocksDB in order to make the LSM-tree function properly. The VLog files also need to be sized accordingly. For database applications, and based on internal testing (discussed later), TRocksDB was most effective in reducing storage fragmentation when the size of the key files (LSM-tree) mirrored the size of the VLog files (1:1 ratio).

An XFS file system was used for tuning TRocksDB as it supports a high-end file limit (or ulimit) of 500,000+ files. Given that 150,000+ files begins to degrade performance, the ulimit[4] was set to 150,000 files, with 80,000 open files used to conduct calculations. It was very important to stay within a 100,000 open file limit as additional files could be needed to run TRocksDB (such as options files, cache files, and level migration files).

In addition to setting the ulimit to 150,000+ files, with 80,000 open files, tuning TRocksDB for performance optimization also requires that other items be addressed, such as the:

- ***Number of CPU threads/cores and DRAM***
- ***Projected key and value sizes***
- ***Number of background jobs and write buffers***
- ***Number of slowdown and stop triggers***
- ***Size of write buffers***
- ***Size of the memory footprint***

All of the calculations and testing presented are based on KIOXIA America system testing with system metrics outlined above, and 80,000 open file handles. Users can invoke performance improvements within TRocksDB using an online Tuning Guide spreadsheet which will output optimized TRocksDB settings based on workload and server specifications. Optimization examples are presented below.

TRocksDB Tuning Options

Within TRocksDB, there are options available that can be tuned through the options file. Some of the options are preset[5] and some of the options need to be left untouched in order for the database engine to function normally. These options are dependent on the system configuration, and the workload (of which key and value sizes are extremely important). For example, a 10-byte key with 10,000-byte value workload will have very different file sizes than a 100-byte key and a 400-byte value workload. The size of the keys will also impact the size of write buffers, as the size of the values will impact the size of the vlog files. The domino effect continues as the size of the cache is also impacted by the memory footprints of the write buffers. And, from a CPU perspective, a server with 128 cores will support a higher number of background jobs and write buffers than a server with 12 processing cores. The TRocksDB system works best when it is tuned to compensate for each of these factors. The recommended tuning options include:

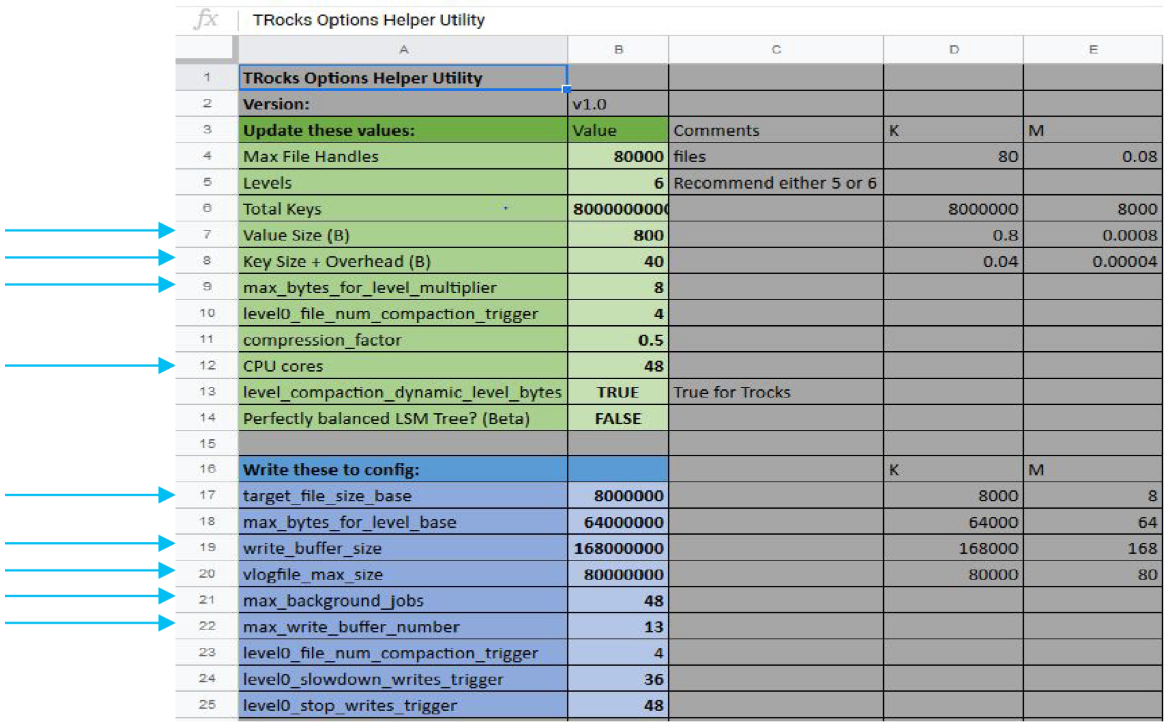
write_buffer_size	target_file_size_base
max_write_buffer_number	max_bytes_for_level_base
max_background_jobs	vlogfile_max_size
num_threads	cache_size

Key Elements Required for Tuning

CPU threads are key elements required to tune TRocksDB and are calculated as 2x the number of cores when hyper-threading or simultaneous multi-threading is enabled. Other tuning parameters include available DRAM, the key and value sizes, and the database size required for the number of usable key/value (k/v) pairs to be projected. For test purposes, the server included 24 cores, 128 gigabyte[6] (GB) DRAM, and a k/v size of 20/800, using an 8 billion key database. The following metrics were input into the Tuning Guide:

Total Keys (projected k/v pairs for database size)	8 billion
Key Size + Overhead	20-byte key with 20-bytes of overhead (for pointer)
Value Size	800 bytes
CPU Cores	48 CPU threads available

Once these metrics were input into the Tuning Guide, the utility updated the tuning options[7]. An example of the actual Tuning Guide output is presented in Figure 1:



fx TRocks Options Helper Utility

	A	B	C	D	E
1	**TRocks Options Helper Utility**				
2	**Version:**	v1.0			
3	**Update these values:**	Value	Comments	K	M
4	Max File Handles	**80000**	files	80	0.08
5	Levels	**6**	Recommend either 5 or 6		
6	Total Keys	**800000000**		8000000	8000
7	Value Size (B)	**800**		0.8	0.0008
8	Key Size + Overhead (B)	**40**		0.04	0.00004
9	max_bytes_for_level_multiplier	**8**			
10	level0_file_num_compaction_trigger	**4**			
11	compression_factor	**0.5**			
12	CPU cores	**48**			
13	level_compaction_dynamic_level_bytes	**TRUE**	True for Trocks		
14	Perfectly balanced LSM Tree? (Beta)	**FALSE**			
15					
16	**Write these to config:**			K	M
17	target_file_size_base	**8000000**		8000	8
18	max_bytes_for_level_base	**64000000**		64000	64
19	write_buffer_size	**168000000**		168000	168
20	vlogfile_max_size	**80000000**		80000	80
21	max_background_jobs	**48**			
22	max_write_buffer_number	**13**			
23	level0_file_num_compaction_trigger	**4**			
24	level0_slowdown_writes_trigger	**36**			
25	level0_stop_writes_trigger	**48**			

Figure 1. Tuning Guide metrics for TRocks 20/800 Database

The spreadsheet recommendations were applied to the TRocksDB options file. When creating the options file from scratch, there is an example located in the rocksdb/tools folder, under trocks_options.ini. When the database is started, it then creates a default 'OPTIONS' file. The options are saved here when the database is stopped, and can be modified before restarting the database. The test scripts in DB_Bench also can specify options that bypass the options file.

Test Procedures and Equipment

KIOXIA America ran two different workloads, covering both the loading of large data sets from scratch and intensive reading in normal operations. Tuning options were based from the results of the online spreadsheet.

LOAD TEST

This test determines the time it takes for TRocksDB to load 4 billion key-value pairs. The workload is 100% sequential writes, starting with a completely empty database, and with results captured in total run time.

Optimized tuning enables TRocksDB to run more efficiently while decreasing the run time of load operations.

RUN TEST

This test determines the amount of new data that TRocksDB could push into a database while running a 48-thread read workload. The workload is read intensive, but with write components.

Optimized tuning enables TRocksDB to read a database more efficiently by enabling more keys to be loaded while the read workload is being performed.

For comparative purposes, three distinct tests were conducted: (1) the optimal setting reported to the spreadsheet; (2) with each setting cut in half; and (3) with each setting doubled.

Test Procedures:

To create a workload that provided testing of TRocksDB in a consistent and efficient manner, while also delivering repeatable results, the DB_Bench tool was selected as the test engine. The DB_Bench utility provides a host of default tests and workloads, but the two selected for optimized tuning include:

- Workload T2: a Load test that represents a 100% sequential write workload
- Workload T5: a Run test that includes a 48-thread read-intensive workload combined with a single-threaded sequential write workload

Test Procedure	Workload T2	Workload T5
Set-up	*The server is cleared of all databases* *The options script is modified to meet the criteria of the specific test*	*The database to be used for T5 is created by running the previous T2 test* *The created database from T2 can be used to read for the T5 test*
Run Database Creation (T2) --- *Run Database Read While Writing (T5)*	*The DB_bench tool is launched via a test script* *The script loads key-value pairs based on the options script*	*The DB_bench tool is launched via a test script* *The script reads and loads key-value pairs based on the options script*
Measurement Process	*For each version of options, the log is exported every 60 seconds* *The measurement logs are saved and analyzed after each test*	*The log is exported every 60 seconds during the entire test run* *The measurement logs are saved and analyzed after each test*
Recording Process	*Total run time and WA* *Latency and data throughput*	*Total run time and WA* *Latency and data throughput*

Test Equipment:

The hardware and software equipment used for the Workload T2 (Load) and Workload T5 (Run) tests included:

- ***Database Tested:*** TRocksDB version t6.4.6.27
- ***Server:*** Tyan AMD® EPYC™ 7601P single socket server featuring 24 processing cores, 2.20 GHz frequency, and 256GB of DDR4 RAM
- ***Operating System:*** Ubuntu® 18.04
- ***Application:*** Databases
- ***Storage Devices:*** Eight (8) NVMe™ SSDs in Linux® RAID0 with 500GB capacities
- ***Benchmark Test Software:*** DB_Bench software (included in the TRocksDB source code)
- ***Configuration Files:*** Set to 'optimized for general-purposes' level of operation and performance

Test Results and Analysis

The tuning results from the DB_Bench tests are divided as follows:

- *CPU Threads Tuning*
- *Background Jobs Tuning*
- *Write Buffer Number Tuning*
- *Target File Size Tuning*
- *Write Buffer Size Tuning*
- *Cache Size Configuration*

<u>CPU Threads Tuning:</u>

The CPU thread count impacts the other five tuning results, so these numbers were calculated first. There are several things that need to be adjusted to match the thread count of all of the CPUs. All modern CPUs are multi-core and multi-threaded, so make sure that you use the threads that are available, and not the core count. As such, the settings that need to be changed are as follows:

Settings	Calculation	Example Setting
max_background_jobs	Set To CPU THREAD Count	48
max_write_buffer_number	(CPU Threads / 4) + 1	13
level0_slowdown_writes_trigger	CPU Threads x 3/4	36
level0_stop_writes_trigger	Set To CPU THREAD Count	48

For this tuning option, we used a single socket, AMD 24-core server that equates to 48 CPU threads that were used for all calculations. The slowdown and stop write triggers were only used for testing. The tests did not demonstrate a stall, so the triggers were never reached.

The write buffer number is an important metric that will be required later in the testing. Given that the level0_file_num_compaction_trigger is always set to 4, when four buffers are full, a compaction is launched and the four buffers are compacted into a single file. The size of these buffers is determined by the size of the keys and values that will be covered later in this section.

<u>Background Jobs Tuning:</u>

The results (Figure 2) show a different number of background jobs configured when running the same workload. On the test server, a 100% write workload was run as well as 48 threads of read and one simultaneous write thread. The T2 write workload run time is shown in yellow, while the T5 read workload run time is shown in blue.

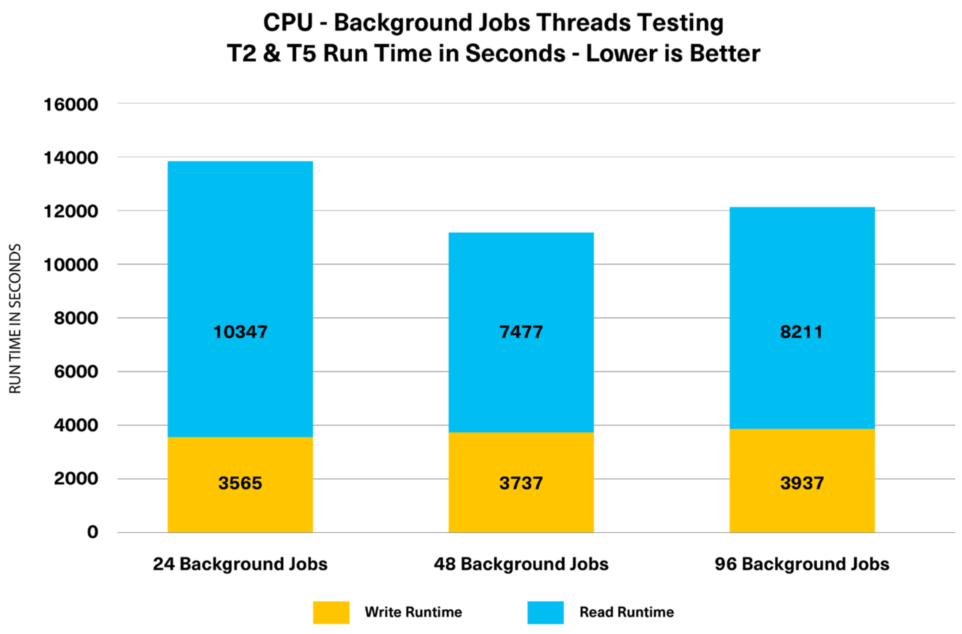


Figure 2: Results of background jobs tuning

The impact that the number of background jobs had on sequential write workload performance was not significant. In fact, from a performance perspective, running 24 background jobs, as well as 96 background jobs was actually slower versus setting the background jobs based on the available CPU threads. Though fewer background jobs showed a small gain in sequential write performance, the corresponding read results, under the same conditions, were 38% worse. ***The optimal setting is when the number of background jobs equals the available CPU threads.***

<u>Write Buffer Number Tuning:</u>

The number of write buffers available to service a workload also has an impact on database performance. This calculation, similar to background jobs tuning, is also based on the number of CPU threads. The memory buffer quantity setting should be equal to one-quarter the thread count, plus 1. When paired with the correct buffer size, this enables compaction from any level to run, while still allowing at least one buffer to be available for writing. ***For this test case, one-quarter of the thread count was equal to 12 plus 1, resulting in a write buffer number of 13.***

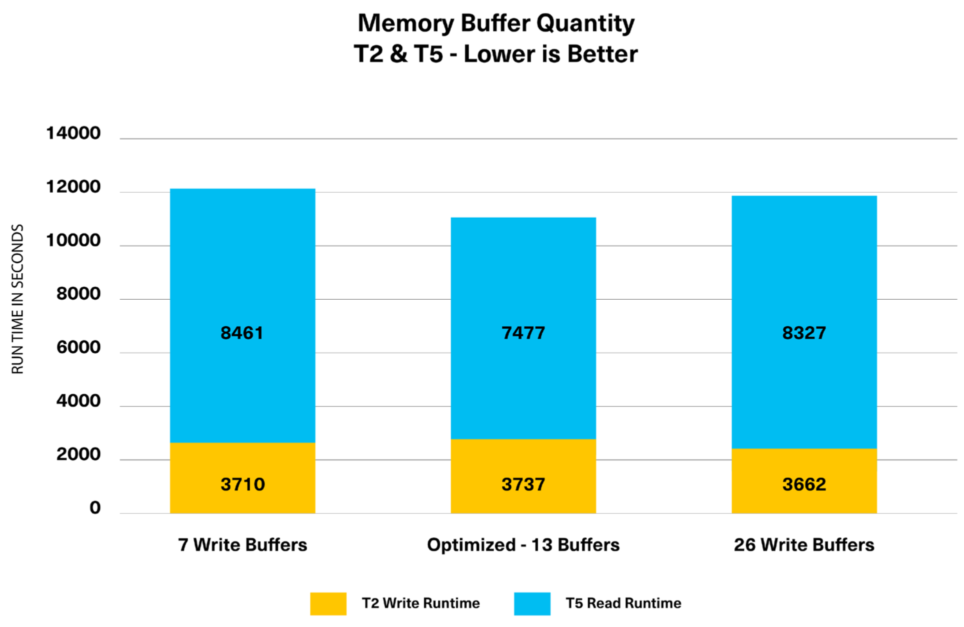


Figure 3: Results of write buffer number tuning

To validate optimal performance using 13 write buffers within the test server (Figure 3), write buffer number tests were also run with 7 write buffers (that use about one-half of the suggested number of write buffers), and 26 write buffers (that use about 2x of the suggested number of write buffers).

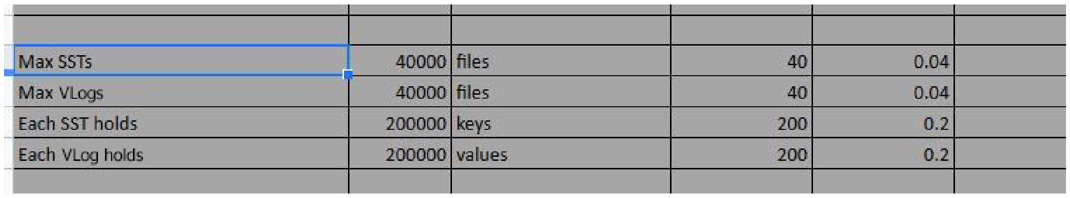
The results indicate that a 'read while writing' operation is most impacted by the write buffer quantity as both higher and lower buffer quantities only showed a small gain in sequential write performance. However, read performance was substantially impacted by both high and low read buffer quantities and ran significantly slower. When the server is busy, the write buffers need to be set to the correct quantity and size to achieve optimal performance. ***For this test case, the write buffers were set to 13 (or ¼ the thread count plus 1).***

<u>Target File Size Tuning:</u>

To achieve optimized TRocksDB performance, the number of CPU threads, background jobs, and write buffers are now calculated and set-up in the Tuning Guide. The next item to address are the size of the buffers that depend on the key and value sizes, and the memory footprint of the server.

Since the keys and values are pulled into memory during compaction and file creation, their respective sizes, as well as the size of the file that they are placed into, deliver large differences in performance. If a database has very large values and is configured for a large number of values per file, then the result will be very large files at the top of the database and very slow compactions since very large values can cause 'out of-memory' conditions. ***Users with value sizes in excess of 10MB may want to deviate from this guide regarding memory buffers and calculate on their own based on the size of the memory footprint on their server.***

For these tests, a key size of 20 bytes and a value size of 800 bytes was used. These k/v sizes are similar to what is used in real world applications, and are the standard testing numbers used by Facebook for RocksDB testing. The database was setup with 80,000 files (open file ulimit), a database size of 8 billion k/v pairs, and a 6-level database. The calculations result in an optimal configuration of 40,000 sorted-string tables[8] (SSTs) and VLog files, with 200,000 keys or values per file (Figure 4).

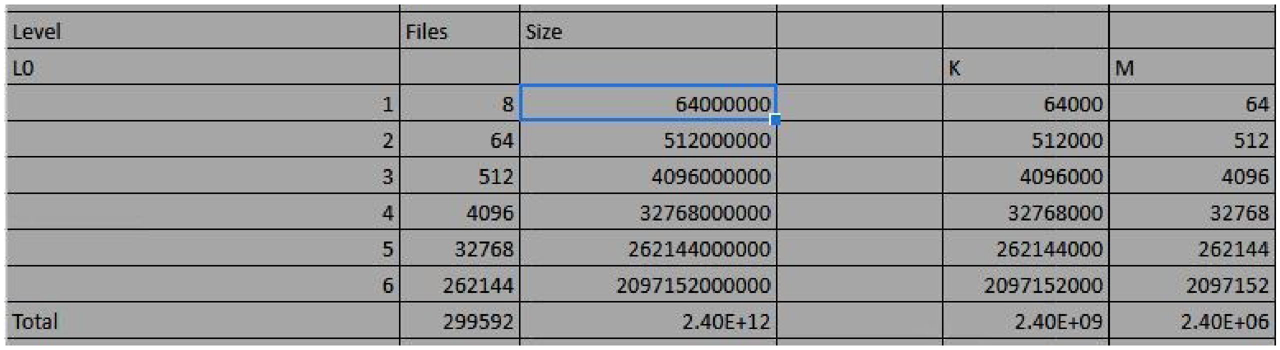


Max SSTs	40000	files	40	0.04	
Max VLogs	40000	files	40	0.04	
Each SST holds	200000	keys	200	0.2	
Each VLog holds	200000	values	200	0.2	

Figure 4: SST and VLog files from the TRocksDB Tuning Guide example

Since the SST files hold the keys, SST file sizes are then calculated based on the optimal number of keys per file, multiplied by the total key size (the actual key size plus the overhead of the VLog pointer). ***This results in an SST file size of 8MB, so the 'target_file_size_base' should be set to 8MB.***

The TRocksDB database was designed around an 8x level multiplier, with L1 being the smallest and L2 being 8x larger (that also represents the compaction scheme). All eight L1 files are compacted into a single L2 file, and new data is written into L1 from L0 (cache). If the SST file size is 8MB, and there are eight files in the level, the level equates to 64MB. The 'max_bytes_for_level_base' should therefore be set to 64MB (or 64,000,000 bytes). Once the 'level base' is set, all other level sizes will automatically be configured in the TRocksDB solution (Figure 5).



Level	Files	Size			
L0				K	M
1	8	64000000		64000	64
2	64	512000000		512000	512
3	512	4096000000		4096000	4096
4	4096	32768000000		32768000	32768
5	32768	262144000000		262144000	262144
6	262144	2097152000000		2097152000	2097152
Total	299592	2.40E+12		2.40E+09	2.40E+06

Figure 5: the SST file LSM-tree output from the TRocksDB Tuning Guide example

To show that this configuration is optimal for a 20/800 database, the tests were run with different target file sizes, with the configured file sizes, as well as larger and smaller file sizes, as follows:

For the T2 sequential write test, the results show that the larger target file size has the best write performance. This is due to having fewer compactions for the number of keys and creating fewer files overall. Larger files do compaction less frequently so the less writing it performs results in a little more efficient use of memory (about a 7% increase in write performance). However, using small files may cause memory thrashing since more frequent compactions of the LSM-tree are required.

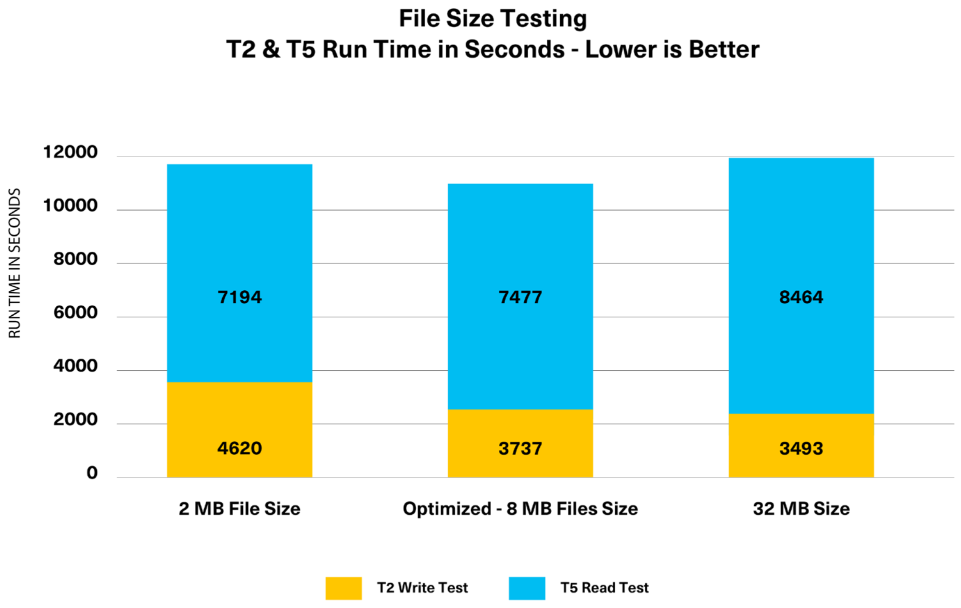


Figure 6: results of read and write testing when larger files are used

For large files, the same key and value size means that there are more keys per file, so as the read threads search the files for keys, there are more keys to sort through that results in slower reads, and the down-side of using larger file sizes. ***Smaller file sizes enable read threads to search and read keys faster, and another way to optimize the database if it's under a heavy read-intensive workload.***

Write Buffer Size Tuning:

Now that the target file sizes have been calculated, the memory footprint can be optimized next. There are two settings for the memory footprint that are used for tuning and include the 'write_buffer_size' and 'cache_size.' When the configuration includes a 20-byte key size and an 800-byte value size, the recommended write buffer size will be 168MB. As the tests demonstrate (Figure 7), a 168MB write buffer size was optimal for both writing and reading, and outperformed smaller buffers (84MB) and larger ones (336MB) under comparable read and write conditions. ***It is imperative that the 'write_buffer_size' setting is correct.***

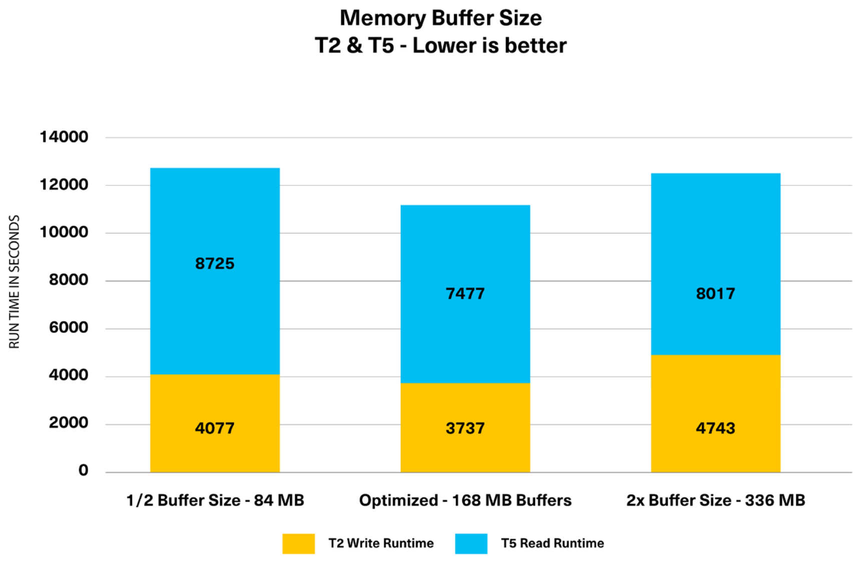


Figure 7: results of the write buffer size tuning

Cache Size Configuration:

The size of the cache is also impacted by the memory footprints of the write buffers and is easy to determine. For testing purposes, a 40GB cache size was used and large enough to cover the test scenarios without causing bottlenecks, but not as large as it could be for optimum usage. The optimal cache size should be comparable to the total system DRAM (the buffer size multiplied by the number of buffers). Relating to TRocksDB performance optimization, the server should have more cache than what is allocated. ***For example, the test case uses 13 write buffers at 168MB each. That equates to 2.1GB of DRAM used for write buffers, leaving ~100GB for DRAM cache.*** The end result is that most of the keys will fit into DRAM, enabling even faster read performance.

Additional Considerations:

To achieve optimized TRocksDB performance, this Tuning Guide should be followed, especially as it relates to the write buffer size and number of write buffers. If the server has a very high CPU thread count (such as a dual 64 core with 256 threads), the standard configuration will have 65 write buffers that could present memory issues relating to large values. In this instance, a value size of 150,000 bytes would yield a write buffer of 30GB, but across 65 write buffers, would require 1,950GB of DRAM (~two terabytes[6]) that could cause ‘out-of-memory’ conditions.

In a database with very large value sizes, such as 3GB, the Tuning Guide recommends that each write buffer size is 600,000GB. The settings for a server under these workloads should disregard the recommended buffer size and hold one file per buffer. ***The Tuning Guide is not recommended for extreme edge configurations.***

Based on internal testing, the value of ‘max_bytes_for_level_base’ should be kept under 100MB. Any setting that is larger than 100MB could result in performance degradation due to compaction. Given this setting, the setting for ‘target_file_size_base’ should be exactly 1/8 that size due to 8x level sizing and ensures that L1 to L2 compaction is always a single compaction that does a complete write of L1 into one L2 file. Since 1/8 of 100MB equals ~12MB, the ‘target_file_size_base’ should never exceed 12MB.

Summary

The ability to tune TRocksDB settings can substantially improve the performance of database workloads based on correct option settings.

To optimize a database for a write-oriented workload:

- *Use a lower number for background jobs*
- *Double the number of write buffers*
- *Use the calculated optimal file sizes for key and value sizes*
- *Consider using a higher ulimit and a high number of files (though this may create significantly more WA, the throughput advantages may make it worth the consideration)*

To optimize a database for a read-oriented workload:

- *Use smaller file sizes with a higher ulimit set for ‘open files’*
- *Increase the cache size to the maximum possible metric based on the write buffer size and buffer capacity*

For extremely large value sizes:

- *Make sure that write memory buffers do not exceed the capacity or else failures and ‘out-of-memory’ conditions’ could occur*
- *As key and value sizes change, the file sizes need to be adjusted accordingly to match the key and value sizes*

There are many ways to tune the TRocksDB database, but using this Tuning Guide can simplify and speed the process.

TRocksDB can be downloaded from the KIOXIA GitHub® site.

Notes:

[1] The specific benefits and performance advantages of TRocksDB when compared to RocksDB can be found in the TRocksDB tech brief entitled, “Introducing the TRocksDB Platform,” November 2019. Source: KIOXIA America.

[2] In a key-value store, records (as well as collection of objects) are stored and retrieved using a key that uniquely identifies the record, while values are records (as well as collection of objects) that are stored for future access and retrieval.

[3] KIOXIA America, Inc. published a Tech Brief in November 2019 entitled, “Introducing the TRocksDB Platform” and downloadable from GitHub. The brief compared performance and write amplification (WA) between RocksDB and TRocksDB database storage engines using four Facebook-developed performance tests and one WA test conducted internally by KIOXIA. Based on the results of these tests, TRocksDB achieved a lower WA for the system as a whole while delivering comparable or better random and sequential performance when tested against the RocksDB platform.

[4] The ulimit controls a number of resources and the ‘ulimit -n’ represents the number of open file descriptors per process.

[5] Adjustments to options that are preset could cause severe performance degradation.

[6] 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 1 GB = 2^{30} bytes = 1,073,741,824 bytes, 1 TB = 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.

[7] In the TRocksDB Tuning Guide, the number of CPU threads is listed as CPU cores and should be the same as the metric for max_background jobs. Also, the cache size is not listed. For testing purposes, a cache size of 40,000,000,000 was used and large enough to cover all of the test scenarios required for this paper without causing bottlenecks, but not as high as it could be for optimum usage. Using a larger cache size will improve read performance, but it is up to the user to determine how much memory needs to be allocated. The typical formula is write_buffer_size * max_write_buffer_number.

[8] Sorted-string tables (SSTs) are files that contain a set of arbitrary, sorted key-value pairs.