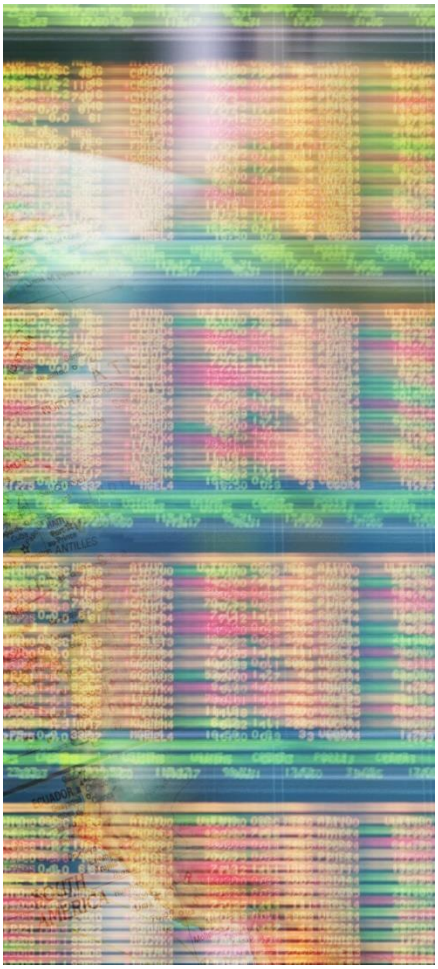
Optimization Guide

Intel® Xeon® Scalable Processor

RocksDB* db_bench Tuning Guide on 3rd Generation Intel® Xeon® Scalable Processors Based Platform



Contents

Revision Record

Date	Rev.	Description
08/15/2021	1.0	Initial Public Release

1. Introduction

This guide is targeted towards users who are already familiar with running db_bench and provides pointers and system setting for hardware and software that will provide the best performance for most situations. However, please note that we rely on the users to carefully consider these settings for their specific scenarios, since db_bench can be configured in multiple ways and this is a reference to one such use-case.

db_bench is the primary tool that is used to benchmark RocksDB* performance. RocksDB inherited db_bench from LevelDB* and enhanced it to support many additional options. Currently, db_bench supports many benchmarks to generate different types of workloads, and its various options can be used to control the tests[1].

The performance of RocksDB is contingent on its tuning. However, due to the large number of configurable parameters, a good configuration is sometimes hard to obtain. RocksDB Advisor is a command-line tool that can be used to assist in finding optimal configuration parameters[2]. We recommend users to use this tool.

RocksDB's performance depends on the tuning of its configuration parameters. Since there is a large number of tunable parameters, we recommend that users use the RocksDB `Advisor` command line tool to determine the optimal configuration parameters[3].

This guide describes mechanism for tuning the following db_bench workloads:

- fillseq
- readrandom
- overwrite
- seekrandom
- readrandomwriterandom
- readwhilewriting

The full list of available workloads can be found at the RocksDB github repo.

3rd Gen Intel® Xeon® Scalable processors deliver industry-leading, workload-optimized platforms with built-in AI acceleration, providing a seamless performance foundation to help speed data's transformative impact, from the multi-cloud to the intelligent edge and back. Improvements of particular interest to this workload applications are:

- Enhanced Performance
- Increased DDR4 Memory Speed & Capacity
- Intel® Advanced Vector Extensions (Intel® AVX), and Intel® AES New Instructions (Intel® AES-NI)

[1] https://github.com/facebook/rocksdb/wiki/Benchmarking-tools

[2] https://rocksdb.org/blog/2018/08/01/rocksdb-tuning-advisor.html

[3] https://github.com/facebook/rocksdb/pull/7240

Tested hardware and software environment for this tuning guide:

Server Configuration	Hardware	Server Platform Name/Brand/Model	Intel® Whitley CRB Wilson City platform
		CPU	Intel® Xeon® PLATINUM 8380 CPU @ 2.30GHz
		BIOS	Intel Corporation
		Memory	8x 32 GB DDR4, 3200 MT/s
		Storage/Disks	4x Intel SSD P4610, 1.6TB
	Software	Operating System	Ubuntu 20.04.2 LTS
		Kernel	5.4.0-70-generic x86_64
		RocksDB	6.15.5

Note: The configuration described in this article is based on 3rd Generation Intel Xeon processor hardware. Server platform, memory, hard drives, network interface cards can be modified based on customer usage requirements.

2. Hardware Tuning

Unless stated otherwise, default settings should be used.

2.1. BIOS Setting

Begin by resetting your BIOS to default setting, then follow the suggestion below for changes:

Configuration Item	Recommended Value
Advanced/Power & Performance/CPU Power and Performance Policy	Performance

Setting the CPU Power and Performance Policy to the recommended value of Performance allows the overall power and performance to be geared towards high performance, even at the expense of energy efficiency.

2.2. Memory Configuration/Settings

It is strongly recommended that at least one DIMM per memory channel is installed. If a channel is left un-populated, it may result in lower CPU-utilization thus reducing overall performance.

2.3. Storage/Disk Configuration/Settings

We recommend using the XFS file system-m. For example, these lines from /etc/fstab will mount nvme devices to previously created filesystem directories:

```
/dev/nvme0n1    /mnt/nvme0   xfs   noatime,nodiratime,nodiscard   0   0
/dev/nvme1n1    /mnt/nvme1   xfs   noatime,nodiratime,nodiscard   0   0
/dev/nvme2n1    /mnt/nvme2   xfs   noatime,nodiratime,nodiscard   0   0
/dev/nvme3n1    /mnt/nvme3   xfs   noatime,nodiratime,nodiscard   0   0
```

We also recommend setting the `read_ahead_kb` kernel parameter to 8 kilobytes. This setting controls how much extra data the kernel reads from disk when performing I/O operations. For example, this could be a shell script set this parameter for four nvme devices:

```
#!/bin/bash
for i in {0..3}; do
    echo deadline > /sys/block/nvme${i}n1/queue/scheduler
    echo 8 > /sys/block/nvme${i}n1/queue/read_ahead_kb
done
```

3. Software Tuning

Software configuration tuning is essential. From the kernel to RocksDB configuration settings, they are all designed for general purpose applications and default settings are almost never tuned for best performance.

3.1. Linux Kernel Optimization Settings

To increase `db_bench` performance, we recommend that users disable swap entirely. To avoid odd performance problems and inconsistencies, we also recommend disabling `zone_reclaim_mode`.

```
$ swapoff --all
$ echo 0 > /proc/sys/vm/zone_reclaim_mode
```

We also recommend setting the CPU frequency scaling governor to performance.

```
#!/bin/bash
for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
do
    [ -f $CPUFREQ ] || continue
    echo -n performance > $CPUFREQ
done
```

RocksDB may open a number of files during a `db_bench` run, and this may sometimes cause a “too many files open” error. To avoid this condition, increase the maximum open files to at least `500000`. Edit /etc/sysctl.conf and append the line: fs.file-max=500000

Then read the parameters from the file with:

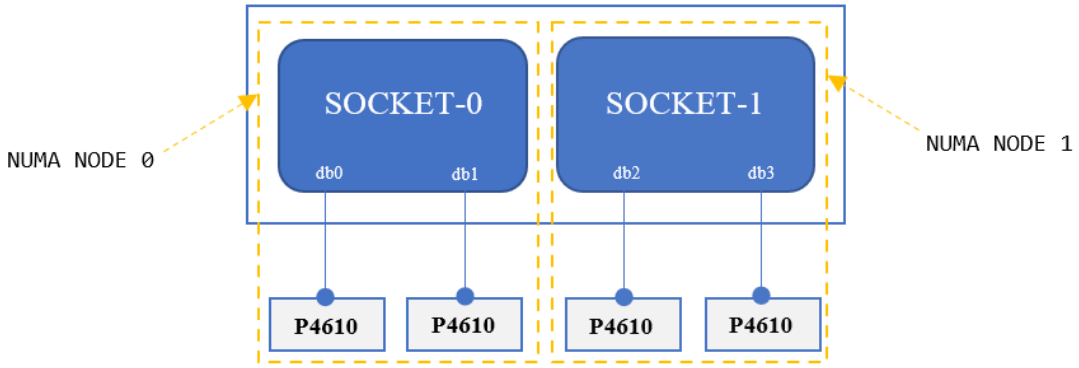
```
$ vi /etc/sysctl.conf
$ fs.file-max=500000
$ sysctl -p
```

3.2. Building RocksDB

RocksDB version 6.15.5 can be downloaded and installed using the following steps. The complete installation guide, including prerequisites, is available on GitHub.

```
$ wget https://github.com/facebook/rocksdb/archive/refs/tags/v6.16.3.tar.gz
$ tar xf v6.16.3.tar.gz
$ cd rocksdb-6.16.3
$  make -j32 release
```

For all the db_bench workloads, we recommend four separate db_bench processes with two processes per socket and each process using its own NVMe drive.



3.3. db_bench Tuning

Sections 3.3.1 to 3.3.6 list scripts that can be used to run the six selected db_bench workloads – fillseq, readrandom, overwrite, seekrandom, readrandomwriterandom, and readwhilewriting. To launch four database instances, a script must be executed four times, each using a different value for NUMA node binding, database directory, and WAL directory. The following table may be used as a guide.

Database instance	Socket	$NUMA_NODE	$DB_DIR	$WAL_DIR
1	0	0	/mnt/nvme0/db0	/mnt/nvme0/wal0
2	0	0	/mnt/nvme1/db1	/mnt/nvme1/wal1
3	1	1	/mnt/nvme2/db2	/mnt/nvme2/wal2
4	1	1	/mnt/nvme3/db3	/mnt/nvme3/wal3

Table 1. Recommended NUMA node binding and database & WAL directory for the four database instances.

We recommend users to run the command below for clearing kernel buffers before running each workload.

```
$ sync; echo 3 > /proc/sys/vm/drop_caches
```

RocksDB uses LSM tree that grows in size as more data is added to a database. To get a consistent performance result, we recommend users to follow a chosen sequence of running the workloads and stick with it. For example, one such sequence can be:

fillseq → readrandom → seekrandom → overwrite → readrandomwriterandom → readwhilewriting

For all the configuration parameters that are recommended for the six workloads, we assume a memory size of 256GB to be available for use. The key and value sizes chosen are 20 and 40 bytes respectively. For a different size of available memory, users are recommended to change the sizes of the write buffer and block cache accordingly.

3.3.1. The fillseq workload

The fillseq workload creates a database and writes a sequence of key-value pairs. The workload is a single-threaded workload and should use the vector memtable. Note that the NUMA_NODE, DB_DIR, and WAL_DIR variables should be set appropriately for each of the four database instances as recommended in Table 1.

```
#!/bin/bash
numactl -m $NUMA_NODE -N $NUMA_NODE $ROCKSDB_HOME/db_bench \
--benchmarks=fillseq \
--use_existing_db=0 \
--db=$DB_DIR \
--wal_dir=$WAL_DIR \
--key_size=20 \
--value_size=400 \
--num=1200000000 \
--threads=1 \
--max_background_jobs=12 \
--block_size=4096 \
--write_buffer_size=1073741824 \
--arena_block_size=16777216 \
--max_write_buffer_number=50 \
--memtablerep=vector \
--allow_concurrent_memtable_write=false \
--cache_size=0 \
--batch_size=4 \
--bloom_bits=10 \
--compression_type=snappy \
--benchmark_write_rate_limit=0 \
--rate_limit_delay_max_milliseconds=1000000 \
--target_file_size_base=1073741824 \
--max_bytes_for_level_base=10737418240 \
--max_bytes_for_level_multiplier=10 \
--level0_file_num_compaction_trigger=10000 \
--level0_slowdown_writes_trigger=1048576000 \
--level0_stop_writes_trigger=1048576000 \
--soft_pending_compaction_bytes_limit=274877906944 \
--hard_pending_compaction_bytes_limit=549755813888 \
--use_direct_reads=0 --use_direct_io_for_flush_and_compaction=0 \
--disable_wal=1 --verify_checksum=1 \
--stats_per_interval=1 --stats_interval_seconds=60 --histogram=1
```

3.3.2. The readrandom workload

The readrandom workload randomly reads data from an existing database. The number of threads, per a database instance, for this workload can be as high as the total number of logical cores available in the system.

```
#!/bin/bash
numactl -m $NUMA_NODE -N $NUMA_NODE $ROCKSDB_HOME/db_bench \
--benchmarks=readrandom \
--disable_auto_compactions=1 \
--use_existing_db=1 \
--db=$DB_DIR \
--wal_dir=$WAL_DIR \
--key_size=20 \
--value_size=400 \
--num=1200000000 \
--duration=480 \
--threads=160 \
--block_size=4096 \
--cache_size=17179869184 \
--cache_numshardbits=10 \
--compression_type=snappy \
--arena_block_size=16777216 \
--memtablerep=skip_list \
--bloom_bits=10 \
--use_direct_reads=0 \
--use_direct_io_for_flush_and_compaction=0 \
--verify_checksum=1 \
--seed=1576170874 \
--stats_per_interval=1 \
--stats_interval_seconds=60 \
--histogram=1
```

3.3.3. The seekrandom workload

The seekrandom workload below executes random seeks followed by one Next call on the underlying iterator. Just like the readrandom workload, the number of threads, per a database instance, for this workload can be as high as the total number of logical cores available in the system.

```
#!/bin/bash
numactl -m $NUMA_NODE -N $NUMA_NODE $ROCKSDB_HOME/db_bench \
--benchmarks=seekrandom \
--disable_auto_compactions=1 \
--use_existing_db=1 \
--db=$DB_DIR \
--wal_dir=$WAL_DIR \
--key_size=20 \
--value_size=400 \
--num=1200000000 \
--duration=480 \
--threads=160 \
--seek_nexts=1 \
--block_size=4096 \
--cache_size=17179869184 \
--cache_numshardbits=10 \
--compression_type=snappy \
--arena_block_size=16777216 \
--memtablerep=skip_list \
--bloom_bits=10 \
--use_direct_reads=0 \
--use_direct_io_for_flush_and_compaction=0 \
--verify_checksum=1 \
--seed=1576170874 \
--stats_per_interval=1 \
--stats_interval_seconds=60 \
--histogram=1
```

3.3.4. The Overwrite workload

The overwrite workload is a write only workload that updates an already existing database. Unlike the readrandom and seekrandom workloads, this workload should use fewer number of threads.

```
#!/bin/bash
numactl -m $NUMA_NODE -N $NUMA_NODE $ROCKSDB_HOME/db_bench \
--benchmarks=overwrite --disable_auto_compactions=1 --use_existing_db=1 \
--db=$DB_DIR --wal_dir=$WAL_DIR \
--key_size=20 \
--value_size=400 \
--num=1200000000 \
--duration=480 \
--threads=16 \
--block_size=4096 \
--cache_size=0 --cache_numshardbits=6 \
--max_write_buffer_number=40 \
--write_buffer_size=1073741824 --arena_block_size=16777216 \
--compression_type=snappy \
--target_file_size_base=1073741824 \
--max_bytes_for_level_base=10737418240 \
--max_bytes_for_level_multiplier=10 \
--memtablerep=skip_list \
--min_write_buffer_number_to_merge=4 \
--max_background_jobs=12 \
--subcompactions=8 \
--bloom_bits=10 \
--use_direct_reads=0 \
--level0_file_num_compaction_trigger=40 \
--level0_slowdown_writes_trigger=10485760 \
--level0_stop_writes_trigger=10485760 \
--use_direct_io_for_flush_and_compaction=0 \
--soft_pending_compaction_bytes_limit=274877906944 \
--hard_pending_compaction_bytes_limit=549755813888 \
--verify_checksum=1 \
--seed=1576170874 \
--stats_per_interval=1 \
--stats_interval_seconds=60 \
--histogram=1
```

3.3.5. The readrandomwriterandom workload

The readrandomwriterandom workload is mix workload where threads do a read (80%) and write (20%) operation. The number of threads, per a database instance, for this workload should not exceed the available number of logical cores in the system.

```
#!/bin/bash
numactl -m $NUMA_NODE -N $NUMA_NODE $ROCKSDB_HOME/db_bench \
--benchmarks=readrandomwriterandom --disable_auto_compactions=1 --use_existing_db=1 \
--db=$DB_DIR --wal_dir=$WAL_DIR \
--key_size=20 \
--value_size=400 \
--num=1200000000 \
--duration=480 \
--threads=160 \
--block_size=4096 \
--cache_size=17179869184 --cache_numshardbits=6 \
--max_write_buffer_number=40 \
--write_buffer_size=1073741824 --arena_block_size=16777216 \
--compression_type=snappy \
--target_file_size_base=1073741824 \
--max_bytes_for_level_base=10737418240 \
--max_bytes_for_level_multiplier=10 \
--memtablerep=skip_list \
--min_write_buffer_number_to_merge=4 \
--max_background_jobs=12 \
--subcompactions=8 \
--bloom_bits=10 \
--level0_file_num_compaction_trigger=40 \
--level0_slowdown_writes_trigger=10485760 \
--level0_stop_writes_trigger=10485760 \
--use_direct_reads=0 \
--use_direct_io_for_flush_and_compaction=0 \
--soft_pending_compaction_bytes_limit=274877906944 \
--hard_pending_compaction_bytes_limit=549755813888 \
--verify_checksum=1 \
--seed=1576170874 \
--stats_per_interval=1 \
--stats_interval_seconds=60 \
--histogram=1
```

3.3.6. The readwhilewriting workload

The readwhilewriting workload uses multiple threads for reads and just one extra thread for writes. Just like the readrandomwriterandom workload, the number of threads, per instance, for this workload should not exceed the available number of logical cores in the system.

```
#!/bin/bash
numactl -m $NUMA_NODE -N $NUMA_NODE $ROCKSDB_HOME/db_bench \
--benchmarks=readrandomwriterandom --disable_auto_compactions=1 --use_existing_db=1 \
--db=$DB_DIR --wal_dir=$WAL_DIR \
--key_size=20 \
--value_size=400 \
--num=1200000000 \
--duration=480 \
--threads=160 \
--block_size=4096 \

--cache_size=17179869184 --cache_numshardbits=6 \
--max_write_buffer_number=40 \
--write_buffer_size=1073741824 --arena_block_size=16777216 \
--compression_type=snappy \
--target_file_size_base=1073741824 \
--max_bytes_for_level_base=10737418240 \
--max_bytes_for_level_multiplier=10 \
--memtablerep=skip_list \
--min_write_buffer_number_to_merge=4 \
--max_background_jobs=12 \
--subcompactions=8 \
--bloom_bits=10 \
--level0_file_num_compaction_trigger=40 \
--level0_slowdown_writes_trigger=10485760 \
--level0_stop_writes_trigger=10485760 \
--use_direct_reads=0 \
--use_direct_io_for_flush_and_compaction=0 \
--soft_pending_compaction_bytes_limit=274877906944 \
--hard_pending_compaction_bytes_limit=549755813888 \
--verify_checksum=1 \
--seed=1576170874 \
--stats_per_interval=1 \
--stats_interval_seconds=60 \
```

4. Encryption in RocksDB

RocksDB currently offers no support for AES-based encryption, but we expect that to change as soon as our PR for AES encryption is merged. The implementation is based on Intel IPP-Crypto library and supports AES-CTR encryption with key sizes of 92, 128, and 256 bits[4].

4.1. Building Intel IPP-Crypto Library

Before building RocksDB with AES-256 support, users need to build and install the IPP-Crypto library.

```
$ wget https://github.com/intel/ipp-crypto/archive/refs/tags/ippcp_2021.2.tar.gz
$ tar xf ippcp_2021.2.tar.gz
$ cd ipp-crypto-ippcp_2021.2
$ CC=gcc CXX=g++ cmake CMakeLists.txt -B_build -DARCH=intel64
$ cd _build
$ make all -j32
$ make install
```

Detailed build instructions are available on GitHub.

4.2. Building RocksDB with AES Support

You may follow the following steps to build RocksDB that has AES support.

```
$ cd /opt
$ git clone --single-branch --branch aes-encryption https://github.com/mulugetam/rocksdb.git
$ cd rocksdb
$ make -j32 release
$ export ROCKSDB_HOME=/opt/rocksdb
$ export PATH=$PATH:$ROCKSDB_HOME
```

4.3. Running db_bench with AES Turned On

To run any of the db_bench workloads described in sections 3.3.1 to 3.3.6 with encryption support, four new configuration parameters need to be added and set.

```
--encrypt_db=1 \                                          # database will be encrypted
--block_cipher=IPP_AES \                                  # AES is the block cipher
--block_cipher_key_size_in_bytes=32 \                     # key size in bytes
--block_cipher_key=a6d2ae2816157e2b3c4fcf098815f7xb \      # the key
```

4.4. Using AES in a RocksDB Application

A RocksDB application can use AES encryption by creating an AES Encryption Provider and setting the database to use an encrypted environment. Example below:

```
DB* db;
Options options;
options.create_if_missing = true;

// create an IPP_AES encryption provider
std::shared_ptr<EncryptionProvider> provider;
Status status = EncryptionProvider::CreateFromString(ConfigOptions(), "IPP_AES", &provider);
assert(status.ok());

// set the key and its size
status = provider->AddCipher("", "a6d2ae2816157e2b3c4fcf098815f7xb", 32, false);
assert(status.ok());

// set new encrypted environment
options.env = NewEncryptedEnv(Env::Default(), provider);

// open database
status = DB::Open(options, kDBPath, &db);
assert(status.ok());
```

Source: https://github.com/mulugetam/rocksdb/blob/aes-encryption/examples/ipp_aes_ctr_example.cc

5. Conclusion

Due to the large number of configuration parameters available in db_bench, the parameter values for the workloads described in sections 3.1.1 to 3.1.6 should be considered as suggestions. Users are highly recommended to run the standard tuning `Advisor` that is distributed with RocksDB.

The performance measurements done with RocksDB demonstrated a significant improvement in throughput and latency when compared to the prior Xeon® generations and other competitive platforms. With 3rd Generation Intel® Xeon® Scalable processor, overall performance further improves as various components, like CPU, memory, and storage work together efficiently for the best user experience.

6. References

[1] RocksDB benchmarking tools: https://github.com/facebook/rocksdb/wiki/Benchmarking-tools

[2] RocksDB Tuning Advisor: https://rocksdb.org/blog/2018/08/01/rocksdb-tuning-advisor.html

[3] Pull Request for AES Encryption: https://github.com/facebook/rocksdb/pull/7240

- Intel® Integrated Performance Primitives Cryptography: https://github.com/intel/ipp-crypto

7. Feedback

We value your feedback. If you have comments (positive or negative) on this guide or are seeking something that is not part of this guide, please reach out and let us know what you think.