PROWESS

Methodology

Behind the Report:

Do Customers Benefit from Intel® In-Memory Analytics Accelerator (Intel® IAA), and at What Cost?

Prowess Consulting, sponsored by AMD, sought to explore the value of using hardware accelerators like Intel IAA to meet the performance and performance-per-watt requirements of in-memory analytics and database workloads versus using lower-cost standard x86 cores that do not require workloads to be modified or any additional processor overhead.

As a starting point, Prowess Consulting explored a claim from the Intel performance index for 4th Gen Intel Xeon Scalable processors.

- 2.01x average performance-per-watt efficiency improvement for RocksDB (with Intel IAA compared to Zstd)[1]

This document provides the system-configuration details and step-by-step procedures that Prowess Consulting used to perform testing of RocksDB workloads to compare performance and efficiency of AMD EPYC™ processors to Intel® Xeon® processors with Intel® In-Memory Analytics Accelerators (Intel® IAA).

The purpose of the testing performed by Prowess Consulting was to evaluate public claims from Intel regarding database performance of systems powered by 4th Gen Intel Xeon Scalable processors with Intel IAA. The claims were used as a starting point for performance and efficiency comparisons against AMD EPYC processors.

Specifically, Prowess Consulting compared two server platforms—one powered by Intel Xeon Platinum 8490H processors configured both with and without Intel IAA, and one configured with AMD EPYC 9734 processors. Both scenarios ran dbench against a RocksDB workload.

The goals were to determine at what point, if any, CPU core utilization for the AMD EPYC 9734 processor–based platform met or exceeded the Intel claim, and to evaluate the overall performance and performance/watt capabilities of the two systems.

Testing was completed on April 19, 2024.

Table 1. System under test (SUT) configurations

	AMD EPYC™ 9734 Processor–Based Configuration	Intel® Xeon® Platinum 8490H Processor–Based Configuration
CPU	AMD EPYC 9734	Intel Xeon Platinum 8490H
Number of CPUs	2	2
Cores/threads per CPU	112/224	60/120
Cores/threads total	224/448	120/240
CPU frequency (all-core/max boost)	3.0/3.0 GHz	2.9/3.5 GHz
Hardware accelerators	N/A	Intel® In-Memory Analytics Accelerator (Intel® IAA)
Installed memory	1.5 TB	1 TB
Memory	64 GB Samsung®	64 GB Micron®
DIMM slots used	24/24	16/16
Operating system (OS)	Ubuntu® 22.04.3 LTS	Ubuntu 22.04.3 LTS
OS kernel	5.15.0-86-generic	5.15.0-91-generic

Intel® In-Memory Analytics Accelerator (Intel® IAA)

Prowess Consulting used RocksDB to compare a system with a 4th Gen Intel Xeon Platinum processor using Intel IAA accelerated compression to the same system using Zstd compression and then to a system with a 4th Gen AMD EPYC processor using Zstd compression. The following procedures assume that the instances have Ubuntu® 22.04 installed and that the instances are connected to a Sentry switched power distribution unit (PDU) for power monitoring. Testing on the Intel processor–powered system was done with 0, 1, and 8 Intel IAA accelerators enabled.

The following section lists the steps needed to set up the instances prior to testing. Default configuration settings were used for the tested servers and environments unless noted in the steps below.

Setting Up the Intel® Xeon® Platinum 8490H Processor–Powered Server with RocksDB

Use the following steps to set up the server powered by an Intel Xeon Platinum 8490H processor:

1. Open a terminal and initiate a Secure Shell (SSH) session to the instance running Ubuntu® 22.04.
2. Run the following commands to update the system and kernel and then reboot:

```
sudo su -
apt update
apt upgrade
apt install linux-generic-hwe-22.04
Reboot
```

3. Run the following command to modify the kernel boot parameters:

```
sudo vim /etc/default/grub
```

   a. Update the GRUB_CMDLINE_LINUX_DEFAULT value to "intel_iommu=on,sm_on no5lvl"

   b. To save and quit, press **<Esc> :wq <Enter>**.

4. Run the following commands to commit the changes and reboot:

```
sudo update-grub
sudo reboot
```

5. Run the following command to install system dependencies:

```
sudo apt-get install -y git cmake ccache python3 ninja-build nasm yasm gawk lsb-release wget
software-properties-common gnupg build-essential autoconf automake autotools-dev libtool pkgconf
asciidoc xmlto uuid-dev libjson-c-dev libkeyutils-dev libz-dev libssl-dev debhelper devscripts
debmake quilt fakeroot lintian asciidoctor file gnupg patch patchutils libgflags-dev liblz4-dev nmon
sysstat
```

6. Run the following command to confirm the ID of the volume that will be used during testing run, henceforth $Volume_ID:

```
fdisk -l
```

7. Run the following command to get the disk UUID, henceforth $Disk_UUID:

```
Ls -lah /dev/disk/by_uuid | grep $Volume_ID
```

8. Run the following command to confirm the NUMA node of the volume:

```
for i in  /sys/class/*/*/device; do pci=$(basename "$(readlink $i)"); if [ -e $i/numa_node ]; then
echo "NUMA Node: `cat $i/numa_node` ($i): `lspci -s $pci`" ; fi; done | sort | grep nvme
```

9. Run the following command to add a file system to the volume:

```
sudo mkfs.xfs /dev/$Volume_ID
```

10. Run the following command to create a directory that will be used for housing the various source code components on the volume to be used in testing, and as a mount point for the test volume, henceforth $Source_Dir:

```
mkdir $Source_Dir
```

11. Run the following command to edit /etc/fstab:

```
sudo vim /etc/fstab
```

    a. Enter a record for the new volume mount point /dev/disk/by-uuid/$Disk_UUID $Source_Dir xfs noatime,nodiratime,nodiscard 0 0

    b. To save and quit, press **<Esc> :wq <Enter>**.

12. Run the following command to mount the volume:

```
sudo mount -a
```

13. Run the following command to change to the $Source_Dir directory you created in step 10:

```
cd $Source_Dir
```

14. Run the following command to clone the Intel® Query Processing Library (Intel® QPL) repository:

```
git clone --recursive https://github.com/intel/qpl.git --branch=v1.4.0
```

15. Run the following commands to build Intel QPL in $QPL_Dir:

```
mkdir -p ./qpl/build
cd qpl/build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=$QPL_Dir ..
cmake --build . --target install
```

16. Run the following command to change back to $Source_Dir:

```
cd $Source_Dir
```

17. Run the following commands to build Intel® oneAPI Threading Building Blocks (Intel® TBB):

```
Wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/af3ad519-4c87-4534-87cb-
5c7bda12754e/l_tbb_oneapi_p_2021.11.0.49527_offline.sh
chmod +x l_tbb_oneapi_p_2021.11.0.49527_offline.sh
./l_tbb_oneapi_p_2021.11.0.49527_offline.sh
```

18. Press **Y** to accept license agreement.
19. Press **N** to not consent to data sharing.
20. Select **Begin installation**.
21. Upon completion, select **Close**.
22. Run the following command to change back to $Source_Dir:

```
cd $Source_Dir
```

23. Run the following command to clone the Intel® Data Accelerator Driver (IDXD) repository:

```
git clone https://github.com/intel/idxd-config.git
```

24. Run the following commands to build IDXD:

```
cd idxd-config
./autogen.sh
./configure CFLAGS='-g -O2'--prefix=/usr --sysconfdir=/etc --libdir=/usr/lib
make
make check
sudo make install
```

25. Run the following command to change back to $Source_Dir:

```
cd $Source_Dir
```

26. Run the following commands to build the Google Test unit test framework (gtest):

```
sudo su -
apt-get install libgtest-dev
cd /usr/src/gtest
cmake CMakeLists.txt
make
cp ./lib/*.a /usr/lib/
```

27. Run the following command to change back to $Source_Dir:

```
cd $Source_Dir
```

28. Run the following command to get the Intel-modified fork of the RocksDB code:

```
git clone --branch pluggable_compression https://github.com/lucagiac81/rocksdb.git
```

29. Run the following command to enter the code directory:

```
cd rocksdb
```

30. Run the following command to pull the Intel IAA compressor code:

```
git clone https://github.com/intel/iaa-plugin-rocksdb.git plugin/iaa_compressor
```

31. Run the following commands to add the Facebook® repository and bring in the corpus pull request:

```
git remote add facebook https://github.com/facebook/rocksdb.git
git fetch facebook pull/10395/head:pull_10395
git merge pull_10395
```

32. Run the following command to create and enter the build directory:

```
mkdir -p ./build; cd ./build
```

33. Run the following command to set the prescribed variables to the Snappy, LZ4, and Zstd libraries' locations:

```
WITH_SNAPPY=/usr/lib/x86_64-linux-gnu/
WITH_LZ4=/usr/lib/x86_64-linux-gnu/
WITH_ZSTD=/usr/lib/x86_64-linux-gnu/
```

34. Run the following command to configure the build:

```
CXXFLAGS="-I/$QPL_Dir/include -I/$QPL_Dir/include/qpl -I/usr/local/include -I/usr/lib64/ -I/usr/lib/"
LDFLAGS="-L$QPL_Dir//lib -L/usr/local/lib/ -L/usr/lib/ -L/opt/intel/oneapi/tbb/latest/lib/intel64/
gcc4.8/ -L/usr/lib x86_64-linux-gnu/ -L$SourceDir/rocksdb/include -DOPT=-DEBUG -O2  -DNDEBUG
-DROCKSDB_ASSERT_STATUS_CHECKED=1 " CFLAGS="-I /usr/local/include" ROCKSDB_CXX_STANDARD="c++17"
DISABLE_WARNING_AS_ERROR=1  cmake .. -DROCKSDB_PLUGINS="iaa_compressor" -DCMAKE_BUILD_TYPE=Release
-DWITH_SNAPPY=$WITH_SNAPPY -DWITH_LZ4=$WITH_LZ4-DWITH_ZSTD=$WITH_ZSTD
```

35. Run the following command to start the build:

```
make install
```

36. Once completed, the RocksDB tools can be found in $SourceDir/rocksdb/build, henceforth $RocksDB_Path.
37. Run the following command to change back to $Source_Dir:

```
cd $Source_Dir
```

38. Run the following command to bring in the Calgary corpus data:

```
wget http://www.data-compression.info/files/corpora/calgarycorpus.zip
```

39. Run the following command to extract the Calgary corpus data:

```
unzip calgarycorpus.zip
```

40. Run the following command to create a folder that will be used to contain the scripts (see the Appendix), henceforth $ScriptsDir:

```
mkdir $ScriptsDir
```

41. Run the following command to change to the $ScriptsDir:

```
cd $ScriptsDir
```

42. Referencing data from the Appendix, create and populate the following files:
    a. $ScriptsDir/power_monitor.sh
    b. $ScriptsDir/RocksRunner-iaa.sh
    c. $ScriptsDir/XeonRocksRunner-zstd.sh
    d. $ScriptsDir/zstdConfig/fillSeqZSTD.sh
    e. $ScriptsDir/zstdConfig/readRandom.sh
    f. $ScriptsDir/zstdConfig/readrandomwriterandom.sh
    g. ScriptsDir/IAAConfig/fillSeqIAA.sh
    h. $ScriptsDir/IAAConfig/readRandom.sh
    i. $ScriptsDir/IAAConfig/readrandomwriterandom.sh
    j. $ScriptsDir/optimizations.sh
43. Run the following command to add the execute permission to each file:

```
chmod +x $filename
```

Setting Up the AMD EPYC™ 9734 Processor–Powered Server with RocksDB

Use the following steps to set up the server powered by an AMD EPYC 9734 processor:

1. Open a terminal and initiate an SSH session to the instance running Ubuntu 22.04.
2. Run the following commands to update the system and kernel and then reboot:

```
sudo su -
apt update
apt upgrade
apt install linux-generic-hwe-22.04
Reboot
```

3. Run the following command to install system dependencies:

```
sudo apt-get install -y libgflags-dev libsnappy-dev zlib1g-dev libbz2-dev  liblz4-dev libzstd-dev
libsystemd-dev  libudev-dev libreadline-dev pkg-config libxml2-dev  libboost-all-dev libelf-dev libnl-
3-dev  build-essential  yasm zlib1g-dev libssl-dev libpci-dev libpcre3 libpcre3-dev cmake
```

4. Run the following command to confirm the ID of the volume that will be used during the testing run, henceforth $Volume_ID:

```
fdisk -l
```

5. Run the following command to get the disk UUID, henceforth $Disk_UUID:

```
Ls -lah /dev/disk/by_uuid | grep $Volume_ID
```

6. Run the following command to confirm the NUMA node of the volume:

```
for i in  /sys/class/*/*/device; do pci=$(basename "$(readlink $i)"); if [ -e $i/numa_node ]; then
echo "NUMA Node: `cat $i/numa_node` ($i): `lspci -s $pci`" ; fi; done | sort | grep nvme
```

7. Run the following command to add a file system to the volume:

```
sudo mkfs.xfs /dev/$Volume_ID
```

8. Run the following command to create a directory that will be used for housing the various source code components on the volume to be used in testing, and as a mount point for the test volume, henceforth $Source_Dir:

```
mkdir $Source_Dir
```

9. Run the following command to edit /etc/fstab:

```
sudo vim /etc/fstab
```

   a. Enter a record for the new volume mount point /dev/disk/by-uuid/$Disk_UUID $Source_Dir xfs noatime,nodiratime,nodiscard 0 0

   b. To save and quit, press **<Esc> :wq <Enter>**.

10. Run the following command to mount the volume:

```
sudo mount -a
```

11. Run the following command to change to the $Source_Dir directory:

```
cd $Source_Dir
```

12. Run the following command to clone RocksDB from the official repository run:

```
git clone https://github.com/facebook/rocksdb.git
```

13. Run the following commands to add the corpus pull request:

```
sudo su -
cd rocksdb
git fetch facebook pull/10395/head:pull_10395
git merge pull_10395
```

14. Run the following commands to set the prescribed variables to the Snappy, LZ4, and Zstd libraries' locations:

```
WITH_SNAPPY=/usr/lib/x86_64-linux-gnu/
WITH_LZ4=/usr/lib/x86_64-linux-gnu/
WITH_ZSTD=/usr/lib/x86_64-linux-gnu/
```

15. Run the following command to configure the build:

```
cmake .. -DWITH_SNAPPY=$WITH_SNAPPY -DWITH_LZ4=$WITH_LZ4  -DWITH_ZSTD=$WITH_ZSTD -DCMAKE_BUILD_
TYPE=Release
```

16. Run the following command to start the build:

```
make install
```

17. Run the following command to bring in the Calgary corpus data:

```
wget http://www.data-compression.info/files/corpora/calgarycorpus.zip
```

18. Run the following command to extract the Calgary corpus data:

```
unzip calgarycorpus.zip
```

19. Referencing data from the Appendix, create and populate the following files:

   a. $ScriptsDir/power_monitor.sh

   b. $ScriptsDir/AMDRocksRunner-zstd.sh

   c. $ScriptsDir/zstdConfig/fillSeqZSTD.sh

   d. $ScriptsDir/zstdConfig/readRandom.sh

   e. $ScriptsDir/zstdConfig/readrandomwriterandom.sh

   f. $ScriptsDir/optimizations.sh

20. Create three copies of the AMDRocksRunner-zstd.sh file.

   a. In file A, set the **ReadCount** to **4** and the **WriteCount** to **8**.

   b. In file B set the **ReadCount** to **6** and the **WriteCount** to **12**.

   c. In file C set the **ReadCount** to **8** and the **WriteCount** to **16**.

21. Run the following command to add the execute permission to each file:

```
chmod +x $filename
```

Running the Tests

Except as noted, the steps below apply to running the test on both the Intel and AMD processor–based platforms. When the RocksRunner*.sh scripts are run, they start system monitoring, and then trigger simultaneous runs of the "readrandom" benchmark. After those steps have completed, a collection of "readrandomwriterandom" benchmarks are also triggered to run simultaneously.

1. If using accelerators for the test run:
    a. Identify the configuration appropriate for the test from $QPL_Dir/share/QPL/configs/. For example, 1n4d1e1w-s-n2.conf.
    b. Run the following command to enable the accelerators:

```
$QPL_Dir/share/QPL/scripts/accel_conf.sh –load $QPL_Dir/share/QPL/configs/$CONFIG_FILE
```

2. Run the following command to populate the Intel IAA test data on the Intel instance with a database name of iaa-dataset:

```
$ScriptsDir/fillSeq-iaa.sh iaa-dataset
```

3. Run the following command to populate the Zstd dataset named "Zstd-dataset" on either instance:

```
$ScriptsDir/fillSeq-zstd.sh zstd-dataset
```

4. To initiate a test run, first reference the appropriate runner script for the test run, henceforth $RunnerScript:
    a. Intel IAA accelerator tests: **RocksRunner-IAA.sh**
    b. Intel Zstd tests: **XeonRocksRunner-zstd.sh**
    c. AMD four-read worker test: **AMDRocksRunner-zstdA.sh**
    d. AMD six-read worker test: **AMDRocksRunner-zstdB.sh**
    e. AMD eight-read worker test: **AMDRocksRunner-zstdC.sh**
5. Identify the database location you will be testing against; that is, $ScriptsDir/iaa-dataset or $ScriptsDir/zstd-data.
6. Run the following command to apply optimizations:

```
$ScriptsDir/optimizations.sh
```

7. Run the following command to start the test:

```
$RunnerScript $numaNode $runID $ConfigDir $DB_path
```

    a. $RunnerScript is the script identified in step 3.
    b. $numaNode is the CPU node on the same channel as the testing volume.
    c. $runID is an identifier under which the results will be stored.
    d. $DB_path is the path identified in step 4.
8. Output of the test run can be found in **/root/results/$RunID**.

Appendix

power_monitor.sh

This script is used to monitor the system power consumption during a test run. It assumes that the system is connected to a Sentry switched PDU for each power supply.

1. Update the $IP1 and $IP2 values to the IPs of the PDUs.
2. Update the $ID value to the ID of the outlet in the PDU.
3. Update the $User and $Pass values as appropriate.

```
pduIP1="$IP1"
pduIP2="$IP2"
result_path=$1
mkdir -p $result_path/p1
mkdir -p $result_path/p2
while true; do
        ts=$(date +%s)
        curl -k https://$USER:$PASS@${pduIP1}/outpower.html?$ID,8 | grep -e "Control State" -e
        "Apparent Power" > $result_path/p1/powerLog1.${ts}
        curl -k https://$USER:$PASS@${pduIP2}/outpower.html?$ID,8 | grep -e "Control State" -e
        "Apparent Power" > $result_path/p2/powerLog2.${ts}
        sleep 2
done
```

RocksRunner-IAA.sh

This script is used to initiate the RocksDB test run and monitoring. Enter the following code for the RocksRunner-IAA.sh file. The CPU values here assume that CPU 1 is 60-core hyper-threaded and that it shares a NUMA channel with the volume used for saving data. Update the CPU IDs if needed.

The UUID of the volume used to save the data will need to replace the $DISK_UUID in the following code. The correct path to the $ScriptsDir will also need to be set.

```
#!/bin/bash
test_source=$3
rid=$2   ### Name (not path)  of directory to save the fillseq data to and pull from in subsequent
tests
node_numa=$1  #### node used to restrict cpu/memory
orig_db=$4
rocks_path="/rocks/Sources/rocksdb-CorpusTest/b3"
ts=$(date +%s)
results_dir="/root/results/${rid}.${ts}"
 alt_node_numa=0
DISK_UUID="$DISK_UUID"
DISK_ID=` ls -lah /dev/disk/by-uuid/ | grep $DISK_UUID | cut -d'/' -f3`
## Run Prep
mkdir -p $results_dir
sync; echo 3 > /proc/sys/vm/drop_caches
numactl -m $alt_node_numa -N $alt_node_numa  nmon -F $results_dir/nmon.out -s2 -c100000 -t &
numactl -m $alt_node_numa -N $alt_node_numa iostat -k -t -o JSON -cdx $DISK_ID 2 > $results_dir/
iostat.out &
numactl -m $alt_node_numa -N $alt_node_numa $ScriptDir/power_monitor.sh $results_dir &
```

```
power_pid=$!
declare -a pids
echo “Starting readRandom Loop at $(date +%s)”  >> $results_dir/progress_summary.txt
for i in $(seq 4); do
        cp -r $orig_db  ${orig_db}_${rid}.${ts}_pass-$i/
        # Start run in background
        case $i in
            1)
                allowedCPU=”60-85”
                ;;
            2)
                allowedCPU=”86-111”
                ;;
            3)
                allowedCPU=”112-119,180-198”
                ;;
            4)
                allowedCPU=”199-224”
                ;;
        esac
        echo “starting run $i with cpu $allowedCPU”
        $test_source/readRandomIAA.sh $rocks_path $allowedCPU  ${orig_db}_${rid}.${ts}_pass-$i >>
       $results_dir/rocks_pass${i}.raw &
        pids[$i]=$!
done
#wait for all to complete
for i in  $(seq 4); do
wait “${pids[$i]}”
done
echo “Completed  readRandom Loop at $(date +%s)”  >> $results_dir/progress_summary.txt
echo “Starting readrandomWriterandom Loop at $(date +%s)”  >> $results_dir/progress_summary.txt
for i in $(seq 8); do
        case $i in
            1)
                allowedCPU=”60-73”
                ;;
            2)
                allowedCPU=”74-87”
                ;;
            3)
                allowedCPU=”88-101”
                ;;
            4)
                allowedCPU=”101-114”
                ;;
            5)
                allowedCPU=”114-119,180-188”
                ;;
```

```
                6)
                    allowedCPU="189-202"
                    ;;
                6)
                    allowedCPU="203-216"
                    ;;
                8)
                    allowedCPU="217-230"
                    ;;
            esac
            $test_source/readrandomwriterandomIAA.sh $rocks_path $allowedCPU  ${orig_db}_${rid}.${ts}_
           pass-$i >> $results_dir/rocks_pass${i}.raw &
            pids[$i]=$!
    done
    #wait for all to complete
    for i in  $(seq 8); do
    wait "${pids[$i]}"
    done
    echo "Completed readrandomWriterandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
    #tally results
    egrep "readrandom|readrandomwriterandom" $results_dir/*.raw >> $results_dir/progress_summary.txt
    ### Monitoring cleanup
    killall nmon
    kill $power_pid
    killall iostat
```

fillSeq-IAA.sh

This script is used to populate RocksDB with Intel IAA test–compatible data.

1. Update $RocksDB_Path to the value on the SUT.
2. Update $TestingVol to point to a path on the volume under test.
3. Update $CorpusPath to point to $SourceDir/calgarycorpus.

```
    #!/bin/bash
    test_dir=$1
    rocks_path=$RocksDB_Path
    mkdir -p /$TestingVol/$test_dir
    numactl -m 1 -N 1 $rocks_path/db_bench --benchmarks=fillseq
     --compression_type=com.intel.iaa_compressor_rocksdb \
     --compressor_options="execution_path=hw" \
     --use_existing_db=0 \
     --db=/rocks/$test_dir \
     --wal_dir=/$TestingVol/wall/$test_dir \
     --key_size=16 \
     --value_size=256 \
     --threads=1 \
     --block_size=4096 \
     --arena_block_size=16777216 \
     --value_src_data_type=file_random \
     --value_src_data_file=$CorpusPath/book1
    done
```

readRandom-IAA.sh

This script is used for the Intel IAA readRandom RocksDB workload. It assumes that the SUT is testing on CPU 1; update the numactl line as needed if testing on an alternate processor node.

```
#!/bin/bash
rocks_path=$1
physcpu=$2
db=$3
numactl -m 1 -N 1 --physcpubind=$physcpu  $rocks_path/db_bench \
  --compression_type=com.intel.iaa_compressor_rocksdb  \
  --compressor_options="execution_path=hw" \
 --benchmarks="readrandom" \
 --threads=25 \
 --use_existing_db \
 --key_size=16 \
 --value_size=256 \
 --db=$db\
 --duration=180
```

readrandomwriterandom-IAA.sh

This script is used to initiate the Intel IAA readRandomWriteRandom RocksDB workload. It assumes that the SUT is testing on CPU 1; update the numactl line as needed if testing on an alternate processor node.

```
#!/bin/bash
rocks_path=$1
physcpu=$2
db=$3
numactl -m 1 -N 1 --physcpubind=$physcpu $rocks_path/db_bench \
  --compression_type=com.intel.iaa_compressor_rocksdb \
  --compressor_options="execution_path=hw" \
  --benchmarks="readrandomwriterandom" \
  --threads=10 \
  --use_existing_db \
  --db=$db\
  --duration=180\
  --key_size=16 \
  --value_size=256 \
  --readwritepercent=80
```

XeonRocksRunner-ZSTD.sh

This script is used to initiate the Zstd compression RocksDB workload and monitoring. The CPU values here assume that CPU 1 is 60-core hyper-threaded and that it shares a NUMA channel with the volume used for saving data. Update the CPU IDs if needed.

The UUID of the volume used to save the data will need to replace the $DISK_UUID in the following code. The correct path to the $ScriptsDir will also need to be set.

```
#!/bin/bash
test_source=$3   #### iaa or lz4 source directory for test scripts
rid=$2
node_numa=$1
orig_db=$4
```

```
rocks_path="$Rocks_DIR"
ts=$(date +%s)
results_dir="/root/results/${rid}.${ts}"
alt_node_numa=0
DISK_UUID="$TESTDISK_UUID"
DISK_ID=` ls -lah /dev/disk/by-uuid/ | grep $DISK_UUID | cut -d'/' -f3`
## Run Prep
mkdir -p $results_dir
mkdir -p $results_dir/commands
sync; echo 3 > /proc/sys/vm/drop_caches
############ Start of monitoring needs to go here
####   numactl -m $alt_node_numa .........
numactl -m $alt_node_numa -N $alt_node_numa  nmon -F $results_dir/nmon.out -s2 -c100000 -t &
numactl -m $alt_node_numa -N $alt_node_numa iostat -k -t -o JSON -cdx $DISK_ID 2 > $results_dir/
iostat.out &
numactl -m $alt_node_numa -N $alt_node_numa $ScriptDir/power_monitor.sh $results_dir &
power_pid=$!
declare -a pids
echo "Starting readRandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
for i in $(seq 4); do
        cp -r $orig_db  ${orig_db}_${rid}.${ts}_pass-$i/
        # Start run in background
        case $i in
            1)
                allowedCPU="60-85"
                ;;
            2)
                allowedCPU="86-111"
                ;;
            3)
                allowedCPU="112-119,180-198"
                ;;
            4)
                allowedCPU="199-224"
                ;;
        esac
        $test_source/readRandomZSTD.sh $rocks_path $allowedCPU  ${orig_db}_${rid}.${ts}_pass-$i >>
       $results_dir/rocks_pass${i}-rr.raw &
        pids[$i]=$!q
done
for i in  $(seq 4); do
        echo "
        waiting for $i  pid ${pids[$i]}
        "
        wait "${pids[$i]}"
done
echo "Completed  readRandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
echo "Starting readrandomWriterandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
```

```
for i in $(seq 8); do
        case $i in
            1)
                allowedCPU="60-73"
                ;;
            2)
                allowedCPU="74-87"
                ;;
            3)
                allowedCPU="88-101"
                ;;
            4)
                allowedCPU="101-114"
                ;;
            5)
                allowedCPU="114-119,180-188"
                ;;
            6)
                allowedCPU="189-202"
                ;;
            7)
                allowedCPU="203-216"
                ;;
            8)
                allowedCPU="217-230"
                ;;
        esac
        $test_source/readrandomwriterandom-zstd.sh $rocks_path $allowedCPU  ${orig_db}_${rid}.${ts}_
       pass-$i >> $results_dir/rocks_pass${i}-rw.raw &
        pids[$i]=$!
done
#wait
for i in  $(seq 8); do
        wait "${pids[$i]}"
done
echo "Completed readrandomWriterandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
egrep "readrandom|readrandomwriterandom" $results_dir/*.raw >> $results_dir/progress_summary.txt
killall nmon
kill $power_pid
killall iostat
```

AMDRocksRunner-ZSTD.sh

This script is used to initiate the Zstd compression RocksDB workload and monitoring on the AMD instance. The CPU values here assume that CPU 0 is 112-core hyper-threaded and that it shares a NUMA channel with the volume used for saving data. Update the CPU IDs if needed.

The UUID of the volume used to save the data will need to replace the $TEST_DISK_UUID in the following code. The correct path to the $ScriptsDir will also need to be set.

This file is used as a basis for creating three different tests.

a. For the four-read workers, eight-write workers configuration, set **ReadCount=4** and **WriteCount=8**.

b. For the six-read workers, twelve-write workers configuration, set **ReadCount=6** and **WriteCount=12**.

c. For the eight-read workers, sixteen-write workers configuration, set **ReadCount=8** and **WriteCount=16**.

```
#!/bin/bash

ReadCount=""
WriteCount=""
test_source=$3   #### iaa or lz4 source directory for test scripts
rid=$2   ### Name (not path)  of directory to save the fillseq data to and pull from in subsequent
tests
node_numa=$1  #### node used to restrict cpu/memory
orig_db=$4
#rocks_path="/rocks/Sources/rocksdb/b4"  ## path to the compiled rocks directory
#rocks_path="/rocks/Sources/rocksdb/build_zstd"
rocks_path=/click/Source/RocksCorpus/rocksdb/build
ts=$(date +%s)
results_dir="/root/IAA_results/${rid}.${ts}"

if [ "$node_numa" -eq 0 ]; then
        alt_node_numa=1
else
        alt_node_numa=0
fi
DISK_UUID="TESTDISK_UUID"
DISK_ID=` ls -lah /dev/disk/by-uuid/ | grep $DISK_UUID | cut -d'/' -f3`
## Run Prep
mkdir -p $results_dir
mkdir -p $results_dir/commands
sync; echo 3 > /proc/sys/vm/drop_caches
numactl -m $alt_node_numa -N $alt_node_numa  nmon -F $results_dir/nmon.out -s2 -c100000 -t &
numactl -m $alt_node_numa -N $alt_node_numa iostat -k -t -o JSON -cdx $DISK_ID 2 > $results_dir/
iostat.out &
numactl -m $alt_node_numa -N $alt_node_numa $ScriptsDir/power_monitor.sh $results_dir &
power_pid=$!
declare -a pids
echo "Starting readRandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
for i in $(seq $ReadCount); do
#copy base data set
        cp -r $orig_db  ${orig_db}_${rid}.${ts}_pass-$i/
        case $i in
            1)
                allowedCPU="1-25"
                ;;
            2)
                allowedCPU="26-50"
                ;;
            3)
```

```
                    allowedCPU="51-75"
                    ;;
                4)
                    allowedCPU="76-100"
                    ;;
                5)
                    allowedCPU="101-111,224-239"
                    ;;
                6)
                    allowedCPU="240-265"
                    ;;
                7)
                    allowedCPU="266-291"
                    ;;
                8)
                    allowedCPU="292-316"
                    ;;
            esac
            echo "using command:   $test_source/readRandomZSTD.sh  $rocks_path $allowedCPU  ${orig_
           db}_${rid}.${ts}_pass-$i >> $results_dir/rocks_pass${i}.raw"
            $test_source/readRandomZSTD.sh  $rocks_path $allowedCPU  ${orig_db}_${rid}.${ts}_pass-$i >>
           $results_dir/rocks_pass${i}.raw &
            pids[$i]=$!
    done
    for i in  $(seq $ReadCount); do
            wait "${pids[$i]}"
    done
    echo "Completed  readRandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
    echo "Starting readrandomWriterandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
    for i in $(seq $WriteCount); do
            case $i in
                1)
                    allowedCPU="1-13"
                    ;;
                2)
                    allowedCPU="14-26"
                    ;;
                3)
                    allowedCPU="27-39"
                    ;;
                4)
                    allowedCPU="40-52"
                    ;;
                5)
                    allowedCPU="53-65"
                    ;;
                6)
                    allowedCPU="66-78"
```

```
                    ;;
                6)
                    allowedCPU="79-91"
                    ;;
                8)
                    allowedCPU="92-104"
                    ;;
                9)
                    allowedCPU="105-111,224-229"
                    ;;
                10)
                    allowedCPU="230-242"
                    ;;
                11)
                    allowedCPU="243-255"
                    ;;
                12)
                    allowedCPU="256-268"
                    ;;
                13)
                    allowedCPU="269-281"
                    ;;
                14)
                    allowedCPU="282-294"
                    ;;
                15)
                    allowedCPU="295-307"
                    ;;
                16)
                    allowedCPU="308-320"
                    ;;
            esac
            $test_source/readrandomwriterandomZSTD.sh $rocks_path $allowedCPU  ${orig_db}_${rid}.${ts}_
           pass-$i >> $results_dir/rocks_pass${i}.raw &
            pids[$i]=$!
    done
    for i in  $(seq $WriteCount); do
        wait "${pids[$i]}"
    done
    echo "Completed readrandomWriterandom Loop at $(date +%s)"  >> $results_dir/progress_summary.txt
    egrep "readrandom|readrandomwriterandom" $results_dir/*.raw >> $results_dir/progress_summary.txt
    killall nmon
    kill $power_pid
    killall iostat
```

fillSeq-ZSTD.sh

This script is used to populate the Zstd data into RocksDB.

1. Update $RocksDB_Path to the value on the SUT.
2. Update $TestingVol to point to a path on the volume under test, such as the $SourceDir.
3. Update $CorpusPath to point to the $SourceDir/calgarycorpus.

This file can be used for the initialization of both the Intel and AMD processor–based instances' Zstd datasets.

```
#!/bin/bash
test_dir=$1
rocks_path=$RocksDB_Path
mkdir -p /$TestingVol/$test_dir
numactl -m 1 -N 1 $rocks_path/db_bench --benchmarks=fillseq  \
 --use_existing_db=0 \
 --db=/$TestingVol/$test_dir \
 --compression_type="zstd" \
 --wal_dir=/$TestingVol/wall/$test_dir \
 --key_size=16 \
 --value_size=256 \
 --threads=1 \
 --block_size=4096 \
 --arena_block_size=16777216 \
 --value_src_data_type=file_random \
 --value_src_data_file=$CorpusPath/book1
```

readRandom-ZSTD.sh

This script is used for both the Intel and AMD processor–based instances' Zstd compression readRandom tests.

```
#!/bin/bash
rocks_path=$1
physcpu=$2
db=$3
#physcpu="0-25"
numactl --physcpubind=$physcpu  $rocks_path/db_bench  \
 --compression_type="zstd" \
 --benchmarks="readrandom" \
 --threads=25 \
 --use_existing_db \
 --db=$db\
 --duration=180 \
 --key_size=16 \
 --value_size=256
```

readrandomwriterandom-ZSTD.sh

This script is used for both the Intel and AMD processor–based instances' Zstd readRandom/WriteRandom compression tests.

```
#!/bin/bash
rocks_path=$1
physcpu=$2
db=$3
numactl -m 0 --physcpubind=$physcpu $rocks_path/db_bench \
```

```
        --compression_type="zstd" \
        --benchmarks="readrandomwriterandom" \
        --threads=10 \
        --use_existing_db \
        --db=$db\
        --key_size=16 \
        --value_size=256 \
        --duration=180\
        --readwritepercent=80
```

Optimizations.sh

This script is used to set disk optimizations. It should be run after rebooting and prior to any testing.

1. Replace $TestingDisk_UUID with the UUID of the disk used in testing.

```
echo 100 > /sys/devices/system/cpu/intel_pstate/min_perf_pct
UUID="$TestingDisk_UUID"
nvme_id=`ls -l /dev/disk/by-uuid/ | grep $UUID | cut -d'>' -f2 | cut -d/ -f3`
echo deadline > /sys/block/$nvme_id/queue/scheduler
echo 8 > /sys/block/$nvme_id/queue/read_ahead_kb
swapoff --all
echo 0 > /proc/sys/vm/zone_reclaim_mode
for eachCPU in `ls /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor`;do
        [ -f $eachCPU ] || continue
        echo -n performance > $eachCPU
done
ulimit -c unlimited
ulimit -n 1000000
```

[1] See [E1] at www.intel.com/performanceindex.