Seagate SSDs:
Linux and MySQL TPC-C Optimizations
Application Note

100813349, Rev. C
April 12, 2017

Rick Stenho, Seagate Sr. Database and Ceph Performance Architect

Revision History

Revision	Date	Description
Rev. C	April 11, 2017	Released for general distribution.
Rev. B	April 6, 2017	Corrected various typo errors.
Rev. A	March 22, 2017	Initial release.

Publication number: 100813349 Rev. C April 2017

Contents

Seagate SSDs: Linux® and MySQL TPC-C Optimizations Application Note

1 Introduction

This application note details procedures for setting up Seagate SAS, SATA and NVMe drives in a database infrastructure to achieve optimum performance. Because each organization and deployment environment is unique, there isn't just one way to configure the Seagate SSDs for best results; there are, however, general guidelines and tools available to assist with the effort. This paper addresses some of them. The discussion is specific to Linux, although most of the popular relational and non-relational databases run on many different OSs and work with Seagate SSDs. The paper also discusses tuning and processes that can be applied to environments to increase TPC-C performance when implementing a range of Seagate SSDs.

The performance-optimization recommendations in this application note center around the following topics:

- Seagate SSD configuration
- Linux OS configuration
- MySQL database configuration

Many of these optimizations are focused on increasing concurrency, decreasing locks and allowing more physical I/Os to the Seagate SSD. The Oracle MySQL 5.6 INNODB database engine was used for all the tests referenced in this paper.

NOTE For simplicity, this paper refers to SAS, SATA and NVMe SSDs as Seagate SSDs.

2 Optimizing Performance

This section describes the following stages, each of which is necessary to optimize the performance of the SSDs:

- Aligning the Seagate SSD
- Tuning the File System
- Tuning the Linux Operating System
- Tuning MySQL

2.1 Aligning the Seagate SSD

The most important step to ensure optimum performance is to create a partition aligned on a specific boundary (such as 4k or 8k). This step allows each read and write to the SSD to require only one physical input/output (I/O) operation. If the Seagate SSD is not partitioned on such a boundary, then reads and writes will span the sector groups, which doubles the I/O latency for each read or write request.

To align the Seagate SSD

1. Identify the Appropriate Seagate SSD Configuration
2. Identify and Choose the Optimum RAID Level
3. Create Aligned Partitions

This section describes each of these steps in detail.

2.1.1 Identify the Appropriate Seagate SSD Configuration

Determine how the SSD will be used in the specific environment. Different SSD usages may require different numbers of SSDs. Will the device be a standalone partition? Part of a logical volume? Part of a RAID group? This section describes the following configuration options:

- Single Seagate SSD for Database Caching
- Multiple Seagate SSDs for Database Caching
- Multiple Seagate SSDs for Persisting Data

2.1.1.1 Single Seagate SSD for Database Caching

When deploying a Seagate SSD for database caching (for example, Facebook's open source Flashcache used in MySQL databases), a single partitioned Seagate SSD is typically suitable if its capacity will meet database requirements at the time of initial deployment and over the subsequent several years. In this case, the `sfdisk` command to create the partition is as follows:

- SAS/SATA SSDs:

```
echo "2048,," | sfdisk -uS /dev/sdX --force
```

- NVMe SSDs:

```
echo "2048,," | sfdisk -uS /dev/nvme#n1 --force
```

2.1.1.2 Multiple Seagate SSDs for Database Caching

When deploying multiple Seagate SSDs for database caching, create a logical volume manager (LVM) over all the Seagate SSDs to simplify administration.

The `sfdisk` command to create a partition for each Seagate SSD is as follows:

- For SAS/SATA SSDs:

```
echo "2048,,8e" | sfdisk -uS /dev/sdX --force
```

- For NVMe SSDs:

```
echo "2048,,8e" | sfdisk -uS /dev/nvme#n1 --force
```

NOTE "8e" is the system partition type for creating a logical volume.

This solution does not require fault tolerance, because the solution is used for write-through caching, meaning data is transparent between disk and cache.

2.1.1.3 Multiple Seagate SSDs for Persisting Data

When deploying Seagate SSDs for persisting data (data written to a non-volatile storage device), use two or more Seagate SSDs to build the RAID array so the storage becomes more fault-tolerant.

The following methods are just two of a number of ways to create a RAID over multiple Seagate SSDs:

- Use LVM with the RAID option.
- Use the software RAID utility MDADM to create the RAID array.

2.1.2 Identify and Choose the Optimum RAID Level

Determine the appropriate RAID level. Whatever your application or environment, follow the practice called stripe and mirror everything (SAME).

To implement SAME

1. Determine Access Type
2. Choose RAID Level

This section describes both of these steps in detail.

2.1.2.1 Determine Access Type

In database deployments, the choice is usually among online transaction processing (OLTP) applications, which can include uses such as airline and hotel reservation systems, corporate financial or enterprise resource planning (ERP) applications, analytical/data warehouse/data mining applications (DW), or a mix of these environments. OLTP applications involve small random reads and writes, as well as many sequential writes for log files. Data warehouse, analytical and data-mining applications involve mostly large sequential reads with very few sequential log writes.

Possible data access types could include the following:

- Small random reads and writes
- Larger sequential reads
- Hybrid (mix of both)

2.1.2.2 Choose RAID Level

Before you set up one or multiple Seagate SSDs in a RAID array—using either LVM on RAID or creating a RAID array using MDADM (multiple device administration)—you should understand not only the system's I/O access pattern, but also your capacity requirements and budget. These requirements dictate the RAID level that will work best for your specific environment.

RAID options include the following:

- RAID 1—Mirroring without striping. Larger investment. Delivers good performance but does not include striping.
- RAID 10—Striping and mirroring. Larger investment. Delivers the best performance
- RAID 5—Striping with parity. Smaller investment but comes with a significant write penalty.

Knowing how to tune the configuration to the application is key to reaping the best performance. Seagate recommends that you implement a RAID 10 array unless budget is a constraint. RAID 5 is best for optimizing performance in a data warehouse/analytics environment, when the majority of the I/Os are reads.

2.1.3 Create Aligned Partitions

Create an aligned partition by using the `sfdisk` command, starting a partition on a 1M boundary (sector 2048). Aligning to a 1M boundary resolves the dependency to align to a 4k, 8k or other boundaries divisible by 4k (for example, 64k and 128k).

For either RAID array, create an aligned partition using `sfdisk` as follows:

- For SAS/SATA SSDs:

```
echo "2048,,fd" | sfdisk -uS /dev/sdX --force
```

- For NVMe SSDs:

```
echo "2048,,fd" | sfdisk -uS /dev/nvme#n1 --force
```

NOTE "fd" is the system identifier for a Linux RAID auto device.

NOTE Creating a partition for LVMs or RAID arrays is not mandatory, as RAW devices can be assigned instead. Align the sectors when you are combining RAW and partitioned devices or when you are simply creating a basic partition. Always create an aligned partition when using a Seagate SSD.

Aligned partitions have now been created and are ready to be used in LVMs or RAID arrays. Instructions for creating these are on the Web or in Linux/UNIX reference manuals. Below are some links that review the process of creating LVM, RAID or LVM on RAID:

- https://raid.wiki.kernel.org/index.php/Partitioning_RAID_/_LVM_on_RAID
- http://www.gagme.com/greg/linux/raid-lvm.php

Remember to specify a stripe width value when creating LVMs with striping or RAID arrays. A 1M stripe width is best as long as the database I/O request is equal to or less than 1M.

2.2 Tuning the File System

Deploying an XFS file system with a 4KB block size resulted in an improvement of 5% to 15% in overall performance. The ext file system is limited to a single mutex per inode, while XFS allows more advanced locking mechanisms, which is important when flushing data from the INNODB buffer. To allocate a 4k block size for XFS, execute the following:

```
mkfs.xfs -s size=4096
```

When considering mount options, you have several options that can be applied to increase performance of the Seagate SSD. For both ext-4 and XFS file systems, the recommendations are as follows:

- For ext4:

  ```
  noatime,nodiratime,max_batch_time=0,nobarrier,discard
  ```

- For XFS:

  ```
  nobarrier,discard,noatime,attr2,delaylog,inode64,noquota
  ```

NOTE The mount option discard could have negative or positive effects on the performance of a system. An alternative to setting the discard option is creating a batch job running the `fstrim` command. This discards unused blocks in the system so performance is only affected when this batch job is run, which would normally be in a maintenance window. Some enterprise environments may not have such a window to run a batch job, so users would benefit by implementing the mount discard option.

2.3 Tuning the Linux Operating System

This section describes various ways to tune the Linux operating system. It includes the following subsections:

2.3.1 Script References

Many variables in the Linux operating system can be tuned to extract the best performance from the Seagate SSDs. Some of these might perform better than others, but when used as a whole, they benefit in more mixed environments. These variables can be set in many different ways, but to persist these variables across system reboots Seagate recommends that you use the script referenced in the next section.

2.3.1.1 Transaction-Based Databases/Applications

For transaction-based databases/applications, the following configuration is recommended:

- For all SAS/SATA SSDs:

```
echo "deadline" > /sys/block/sdX/queue/scheduler
echo 2048 > /sys/block/sdX/queue/nr_requests
echo 1024 > /sys/block/sdX/queue/max_sectors_kb
echo 1024 > /sys/block/sdX/device/queue_depth
echo 0 > /sys/block/sdX/queue/nomerges
echo 0 > /sys/block/sdX/queue/rotational
blockdev --setra 0 /dev/sdX
echo 0 > /sys/block/sdX/queue/add_random
echo 2 > /sys/block/sdX/queue/rq_affinity
echo 1 > /sys/block/sdk/queue/iosched/fifo_batch
echo 0 > /sys/block/sdk/queue/iosched/front_merges
```

```
echo 5 > /sys/block/sdk/queue/iosched/writes_starved
```

- For NVMe SSDs with kernels without BLK-MQ support:

```
echo "deadline" > /sys/block/nvme#n1/queue/scheduler
echo 2048 > /sys/block/nvme#n1/queue/nr_requests
echo 1024 > /sys/block/nvme#n1/queue/max_sectors_kb
echo 1024 > /sys/block/nvme#n1/device/queue_depth
echo 0 > /sys/block/nvme#n1/queue/nomerges
echo 0 > /sys/block/nvme#n1/queue/rotational
blockdev --setra 4096 /dev/nvme#n1
echo 0 > /sys/block/nvme#n1/queue/add_random
echo 1 > /sys/block/nvme#n1/queue/rq_affinity
echo 1 > /sys/block/nvme#n1/queue/iosched/fifo_batch
echo 0 > /sys/block/nvme#n1/queue/iosched/front_merges
echo 5 > /sys/block/nvme#n1/queue/iosched/writes_starved
```

- For NVMe SSDs with kernel BLK-MQ support:

```
echo 0 > /sys/block/nvme#n1/queue/nomerges
echo 0 > /sys/block/nvme#n1/queue/rotational
blockdev --setra 4096 /dev/nvme#n1
echo 0 > /sys/block/nvme#n1/queue/add_random
echo 1 > /sys/block/nvme#n1/queue/rq_affinity
```

2.3.1.2 Data Warehouse and Analytics

For data warehouse or data analytics types of applications/databases, Seagate recommends the following:

- For SAS/SATA SSDs:

```
blockdev --setra 4096 /dev/sdX
```

- For NVMe SSDs:

```
blockdev --setra 8192 /dev/nvme#n1
```

- Set swapiness to 10:
  - To set in a non-persistent value: `sysctl -w vm.swappiness=10`
  - To store in a new persistent value add: `vm.swappiness=10 to the /etc/sysctl.conf file`

2.3.2 Invoking JEMALLOC

JEMALLOC is a general purpose memory allocator that emphasizes fragmentation avoidance and provides better scalable concurrency support. JEMALLOC is normally used in demanding applications, such as MySQL databases.

To invoke the JEMALLOC memory allocator instead of using the default memory allocator from glibc

1. Download and install JEMALLOC for the correct Linux release and version.
2. Add the following environment variable to the .bash_profile:

```
LD_PRELOAD=/usr/lib64/libjemalloc.so.1
```

   NOTE File location could be different based on OS and release

3. Reload variables: `". .bash_profile"`
4. Restart MySQL. After restart, MySQL will use JEMALLOC.
5. To verify if MySQL is using JEMALLOC:

```
ldd /usr/sbin/mysqld
      linux-vdso.so.1 =>  (0x00007fffe79ff000)
      /usr/lib64/libjemalloc.so.1 (0x00007f3e1bd73000)
      libpthread.so.0 => /lib64/libpthread.so.0 (0x000000372f000000)
```

```
        libaio.so.1 => /lib64/libaio.so.1 (0x000000372e400000)
        librt.so.1 => /lib64/librt.so.1 (0x000000372f800000)
        libcrypt.so.1 => /lib64/libcrypt.so.1 (0x000000373e400000)
              libdl.so.2 => /lib64/libdl.so.2 (0x000000372ec00000)
              libstdc++.so.6 => /usr/lib64/libstdc++.so.6
(0x0000003734000000)
              libm.so.6 => /lib64/libm.so.6 (0x000000372f400000)
              libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x0000003732400000)
              libc.so.6 => /lib64/libc.so.6 (0x000000372e800000)
              /lib64/ld-linux-x86-64.so.2 (0x000000372e000000)
              libfreebl3.so => /lib64/libfreebl3.so (0x000000373ee00000)
```

2.3.3 Invoking HugePages

Although Linux uses 4k memory pages, Linux and MySQL can be configured to use HugePages, which are 2M in size. Using HugePages decreases the number of memory pages to 1 from 500, allowing Linux to operate more efficiently.

To check if the system is set up for HugePages, execute the following:

```
cat /proc/meminfo | grep Huge*
```

If any of the values are greater than 0, then HugePages has been enabled.

Next, determine if the number of HugePages is large enough for MySQL.

To setup HugePages for MySQL, calculate the amount of memory MySQL is using, which includes all of its buffers and memory pools. To calculate the memory allocation that MySQL is taking up, in MySQL, execute the following:

```
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
SHOW VARIABLES LIKE 'innodb_additional_mem_pool_size';?
SHOW VARIABLES LIKE 'innodb_log_buffer_size';?
SHOW VARIABLES LIKE 'thread_stack';?
SET @k_bytes = 1024;
SET @m_bytes = @k_bytes * 1024;
SET @g_bytes = @m_bytes * 1024;?
SET @innodb_buffer_pool_size = 2 * @g_bytes;
SET @innodb_additional_mem_pool_size = 16 * @m_bytes;
SET @innodb_log_buffer_size = 8 * @m_bytes;
SET @thread_stack = 192 * @k_bytes;
SELECT (@@key_buffer_size + @@query_cache_size + @@tmp_table_size +
@innodb_buffer_pool_size + @innodb_additional_mem_pool_size +
@innodb_log_buffer_size + @@max_connections * (@@read_buffer_size +
@@read_rnd_buffer_size + @@sort_buffer_size + @@join_buffer_size +
@@binlog_cache_size + @thread_stack)) / @g_bytes AS MAX_MEMORY_GB;

+---------------+
| MAX_MEMORY_GB |
+---------------+
|      17.76384 |
+---------------+
```

To set the number of pages to be used, take the memory needed for all of MySQL and divide that by 2M. For example, the MySQL setup that was used allocated 16G for the buffer and another 1+G for the other buffers/pools. For example, 18G in HugePages was allocated by executing the following commands:

```
echo 9000 > /proc/sys/vm/nr_hugepages
```

Modify /etc/sysctl.conf to add or modify the vm.nr_hugepages settings:

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
vm.nr_hugepages=9000
```

Reboot the server or execute "`sysctl -p`" for the setting to take place.

```
# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.

echo 102 > /proc/sys/vm/hugetlb_shm_group

# Increase the amount of shmem permitted per segment
# (18G in this case).

echo 18874368000 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory.  The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.

echo 4194304 > /proc/sys/kernel/shmall
```

Modify my.cnf to enable HugePages. Add "large-pages" in the mysqld section.

Modify /etc/security/limits.conf to set the MySQL user memlock to unlimited:

```
@mysql   soft    memlock unlimited
@mysql   hard    memlock unlimited
```

Modify the mysqld_safe script. Add "ulimit -l unlimited" in the beginning.

Start MySQL: "`mysqld_safe &`"

Verify that MySQL is using HugePages:

```
cat /proc/meminfo |grep HugePages
AnonHugePages:   4126720 kB
HugePages_Total:    9000
HugePages_Free:     1136
HugePages_Rsvd:        7
HugePages_Surp:        0
```

This confirms that almost 8000 HugePages are being used in this Linux/MySQL environment.

2.3.4 Making the Linux Environment Variables Persist for Seagate SSDs

This section contains the following subsections:

- Seagate SAS/SATA SSDs
- Seagate NVMe SSDs

2.3.4.1 Seagate SAS/SATA SSDs

Device assignments in a Linux server can sometimes change after reboots. On some occasions, for example, a device might be assigned to /dev/sda, while on others it might be assigned to /dev/sdd or any device name. This variability can wreak havoc when modifying the Linux environment variables. To avoid this issue, assignments utilizing the SCSI address should be used so all of the Linux performance variables are persisted properly across reboots.

NOTE When using a file system, use the device UUID address in the mount statement in /etc/fstab so that the mount command is persisted across reboots. To locate the UUID address for each storage device, use 'blkid'.

For single SAS/SATA SSDs, the first step to solve the OS assignments issue is to use the following script, which can be copied and pasted into /etc/rc.local (with the exception of SCSI address of the Seagate SSD device, which is required before executing the script).

The SCSI address in the script (highlighted in bold) must be modified with the address of the Seagate SAS/SATA SSD. To get this value, issue the following command:

```
ls -al /dev/disk/by-id
```

When the Seagate SAS/SATA SSD is installed, Linux assigns a name to the device. For example, the device name can be listed as /dev/sdX, where *X* can be any letter. The output from the `'ls'` command above shows the SCSI address for this Seagate SAS/SATA SSD. Do not use the address that has '-partX' in it.

NOTE Be sure to note this SCSI address, as it is required to create the script below.

NOTE Include one space after the "scsi-" address before the single quote.

Copy the code below and create a file called "ssd_getdevice.sh" with the modification of the SCSI address (found above) that is marked in bold.

```
ssd_getdevice.sh
      ls -al /dev/disk/by-id |grep 'scsi-35000c500178e3a2f ' |grep /sd >
      ssddevice.txt
      awk '{split($11,arr,"/"); print arr[3]}' ssddevice.txt > ssd1device.txt
      variable1=$(cat ssd1device.txt)
      echo "4096" > /sys/block/$variable1/queue/nr_requests
      echo "512" > /sys/block/$variable1/device/queue_depth
      echo "deadline" > /sys/block/$variable1/queue/scheduler
      echo "2" > /sys/block/$variable1/queue/rq_affinity
      echo 0 > /sys/block/$variable1/queue/rotational
      echo 0 > /sys/block/$variable1/queue/add_random
      echo 1024 > /sys/block/$variable1/queue/max_sectors_kb
      echo 0 > /sys/block/$variable1/queue/nomerges
      blockdev --setra 0 /dev/$variable1
      echo 1 > /sys/block/$variable1/queue/iosched/fifo_batch
      echo 0 > /sys/block/$variable1/queue/iosched/front_merges
      echo 5 > /sys/block/$variable1/queue/iosched/writes_starved
```

After saving this file, change the permission of the file to "execute", then place the following command in the /etc/rc.local file:

```
/path/ssd_getdevice.sh
```

To test this script, execute it on the command line exactly as it is stated it in the rc.local file. The next time the system is rebooted, the settings will be set to the appropriate device.

For multiple Seagate SAS/SATA SSDs in the server, the easiest way to address the OS-assignment issue would be to create the script below, which grabs all the SCSI devices (without their partitions) and creates a file that contains all of these devices. If the ssd2device.txt file contains all the SAS/SATA SSD devices you want to configure, then execute the commands from the "while" statement through the "done" command. If there are SCSI devices that you don't want to configure, then edit the ssd2device.txt file, remove the statements, save, then run the script below from the "while" command through the "done" command.

```
ssd_getdevice.sh
      ls -al /dev/disk/by-id |grep 'scsi' |grep /sd > ssddevice.txt; cat
      ssddevice.txt |grep -vE "(part)" > ssd1devices.txt; awk
      '{split($11,arr,"/"); print arr[3]}' ssd1devices.txt > ssd2device.txt;
      while read p; do
      echo "4096" > /sys/block/$p/queue/nr_requests
      echo "512" > /sys/block/$p/device/queue_depth
      echo "deadline" > /sys/block/$p/queue/scheduler
      echo "2" > /sys/block/$p/queue/rq_affinity
      echo 0 > /sys/block/$p/queue/rotational
      echo 0 > /sys/block/$p/queue/add_random
      echo 1024 > /sys/block/$p/queue/max_sectors_kb
      echo 0 > /sys/block/$p/queue/nomerges
      blockdev --setra 0 /dev/$p
      echo 1 > /sys/block/$p/queue/iosched/fifo_batch
      echo 0 > /sys/block/$p/queue/iosched/front_merges
      echo 5 > /sys/block/$p/queue/iosched/writes_starved
      done < ssd2device.txt
```

2.3.4.2 Seagate NVMe SSDs

When a Seagate NVMe SSD is installed, Linux assigns a name to the device. The device name can be listed as /dev/nvme#n1, for example, where # can be any number. The output from the 'blkid' command shows the NVMe device name (do not use the entry that has 'p#' as a suffix). The scripts below generate and execute the Linux commands to configure all of the NVMe devices for optimal performance in a database environment. If you do not want to modify some of the NVMe devices, then execute the top four commands of the appropriate script, edit 'nvm3devices.txt' and remove the devices you don't want to set, and save the file. Then execute the commands from the "while" command to the "done" command.

This section contains the following scripts:

- nvme_getdevice.sh for kernels without BLK-MQ support
- nvme_getdevice.sh for kernels with BLK-MQ support

Copy the appropriate code and create a file called "nvme_getdevice.sh".

nvme_getdevice.sh for kernels without BLK-MQ support:

```
      blkid |grep nvme > nvmedevices.txt
      cat nvmedevices.txt |grep -vE "(p)" > nvme1devices.txt
      awk '{split($0,arr,"/"); print arr[3]}' nvme1devices.txt > nvm2devices.txt
      awk '{split($0,arr,":"); print arr[1]}' nvm2devices.txt > nvm3devices.txt
      while read p; do
      echo "4096" > /sys/block/$p/queue/nr_requests;
      echo "512" > /sys/block/$p/device/queue_depth;
      echo "deadline" > /sys/block/$p/queue/scheduler;
      echo "2" > /sys/block/$p/queue/rq_affinity;
      echo 0 > /sys/block/$p/queue/rotational;
      echo 0 > /sys/block/$p/queue/add_random;
      echo 1024 > /sys/block/$p/queue/max_sectors_kb;
```

```
echo 0 > /sys/block/$p/queue/nomerges;
blockdev --setra 4096 /dev/$p;
echo 1 > /sys/block/$p/queue/iosched/fifo_batch;
echo 0 > /sys/block/$p/queue/iosched/front_merges;
echo 5 > /sys/block/$p/queue/iosched/writes_starved;
done < nvm3devices.txt;
```

nvme_getdevice.sh for kernels with BLK-MQ support:

```
blkid |grep nvme > nvmedevices.txt
cat nvmedevices.txt |grep -vE "(p)" > nvme1devices.txt
awk '{split($0,arr,"/"); print arr[3]}' nvme1devices.txt > nvm2devices.txt
awk '{split($0,arr,":"); print arr[1]}' nvm2devices.txt > nvm3devices.txt
while read p; do
echo "1" > /sys/block/$p/queue/rq_affinity;
echo 0 > /sys/block/$p/queue/rotational;
echo 0 > /sys/block/$p/queue/add_random;
echo 0 > /sys/block/$p/queue/nomerges;
blockdev --setra 4096 /dev/$p;
done < nvm3devices.txt;
```

After saving this file, change permission of the file to "execute," place this command in the /etc/rc.local file:

```
/path/nvme_getdevice.sh
```

To test this script, execute it on the command line exactly how it is stated it in the rc.local file. The next time the system is rebooted, the settings will be set to the appropriate device.

2.4 Tuning MySQL

After aligning the SSDs and tuning the file and operating systems, the next logical step would be to apply tuning settings to the MySQL database itself. Many possible database variables can be set to configure a MySQL database. The options for setting an OLTP database, for example, can be quite different from the tuning options for setting a data warehouse/analytics database. In performing benchmarks for OLTP types, the MySQL database had the following tuning settings applied to achieve optimal performance while using the INNODB database engine.

Recommendations for Seagate SSDs, include:

- innodb_log_file_size = 4G
- innodb_io_capacity = 200000
- default_storage_engine = InnoDB
- innodb_flush_method = O_DIRECT
- innodb_buffer_pool_size = 80% of RAM
- innodb_use_native_aio = ON

Set the Linux variable TMPDIR to a temporary directory for mysql. The default is to use /tmp or the root disk, which could fill the root disk. Assign MySQL TMP directory to another non-root volume inside .bash_profile:

export TMPDIR=/u02/tempdir

(restart MySQL)

3 Conclusion

Implementing offerings in the Seagate SSD portfolio into a MySQL database infrastructure can dramatically increase database performance. By following the simple tuning tips outlined in this document, you can achieve a successful and effective implementation of any Seagate SSD with optimal performance in most environments.

Seagate Technology LLC

AMERICAS Seagate Technology LLC 10200 South De Anza Boulevard, Cupertino, California 95014, United States, 408-658-1000

ASIA/PACIFIC Seagate Singapore International Headquarters Pte. Ltd. 7000 Ang Mo Kio Avenue 5, Singapore 569877, 65-6485-3888 EUROPE, MIDDLE EAST AND AFRICA Seagate Technology SAS 16-18 rue du Dôme, 92100 Boulogne-Billancourt, France, 33 1-4186 10 00

Document Number: 100813349, Rev. C
April 2017