MySQL TPC-C Optimization

Technology Paper

Authored by Rick Stehno,
Oracle DBA Engineer, Seagate

Implementing the Nytro Accelerator Card to Increase Database Performance

Implementing the Seagate® Nytro™ Accelerator Card into a MySQL® database infrastructure can dramatically increase database performance. By following these simple tuning tips outlined in this paper, a successful implementation of the Nytro flash-based card with optimal performance can occur in most environments.

The Nytro Application Acceleration Card is most often purchased to increase one's performance of their database, such as the MySQL™ database on the Linux platform. Once the Nytro card has been installed, the most logical next question would be: "How do I set up the infrastructure in order to get the best performance?" As each organization and environment is unique, no cookie cutter questions will reveal specifically how the Nytro card should be configured for optimum performance, but there are tools available and best practices to assist with this effort.

Most of the popular relational and non-relational databases run on many different operating systems. This guide will describe Linux-specific configurations, but most other operating systems should also work with the Nytro card.

This document will present tuning and processes that can be applied to increase TPC-C performance when using the Nytro card. The performance optimization recommendations that this document will review includes the following:

- Nytro PCIe flash-based card configuration
- Linux operating system configuration
- MySQL database configuration

Many of these optimizations were centered on increasing concurrency, decreasing locks, and allowing more physical I/O to the Nytro card. The Oracle MySQL 5.6 InnoDB storage engine was used for all the tests described in this paper.

Steps to Optimizing Performance

Aligning the Nytro Accelerator Card

To create an aligned partition, use the `sfdisk` command to start a partition on a 1M boundary (sector 2048). Aligning to a 1M boundary resolves the dependency to align to a 4k, 8k, and other boundaries divisible by 4k (for example: 64k, 128k).

Prior to this step, several questions should be posed about your specific deployment and how you are going to use this device. Will this be a standalone partition, part of a logical volume, or part of a RAID group?

Identifying the Appropriate Nytro Accelerator Card

Typically when deploying the Nytro card for database caching (for example, using the Open Source Flashcache developed by Facebook with MySQL), a single-partitioned Nytro card would be suitable if the capacity is sufficient to meet the needs of the database now and over the next several years. In this case, the `sfdisk` command to create the partition would be:

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

If a need exists to deploy multiple Nytro cards for database caching, create a logical volume manager (LVM) over all the Nytro cards to simplify administration. The `sfdisk` command to create a partition for each Nytro card is:

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

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

Neither of these solutions requires fault tolerance since they are used for write-thru caching, meaning data will be transparent between disk and cache.

If the Nytro card is used for persisting data or with write back caching, then multiple Nytro cards are needed to provide fault tolerance. By using two or more Nytro cards to build the RAID array, this concept eliminates any single point of failure. There are a number of ways to create a RAID over multiple Nytro cards, two of which are:

- Use LVM with the RAID option.
- Use the software RAID utility multiple device administration (MDADM) to create the RAID array.

Identifying the Best RAID Level

Oracle coined the term *stripe and mirror everything* (SAME) in 1999 and popularized the practice which many database administrators (DBA) and storage administrators continue to follow.

To implement SAME, first determine how the Nytro cards will be accessed. This could include:

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

In database deployments, your choice is usually among online transaction processing (OLTP) applications such as airline and hotel reservation systems, corporate financial or enterprise resource planning (ERP) applications, or an 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/data mining applications involve mostly large sequential reads with very few sequential log writes.

Before setting up one or many Nytro cards in a RAID array, either using LVM on RAID or creating a RAID array using MDADM, it is important to understand the access pattern of the I/O, capacity requirements and budget. These requirements will dictate which RAID level will work best for the specific environment.

RAID options would include either a RAID 1/RAID 10 configuration (mirroring without striping, or striping and mirroring respectively), or RAID 5 (striping with parity). RAID 1/RAID 10 is a larger investment, but delivers the best performance, whereas RAID 5 costs less but imposes a significant write penalty. To optimize performance of an OLTP application, it is recommended to either implement a RAID 1 or RAID 10 array. If budget is a constraint, then RAID 5 should be considered. In a Data Warehouse/Analytics environment where the majority of the I/O is reads, RAID 5 would be the option to implement. Knowing how to tune the configuration to the application is a key to reaping the best performance.

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

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

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

Keep in mind that it is not mandatory to create a partition for LVMs or RAID arrays. Instead, RAW devices can be assigned. It's important to remember to align the sectors when combining RAW and partitioned devices, or when just creating a basic partition. It's sound practice to always create an aligned partition when using the Nytro card.

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 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

Also, remember that when creating LVMs with striping or RAID arrays, it is important to specify a stripe width value. For years, the recommendation of using a 1M stripe width performed best with both full-table scans and small random I/O to prevent hot disk issues.

File System Tuning

There are many different filesystems to use for a MySQL database. Some perform better in certain cases while the same filesystem might perform less well in others. In-house testing using your equipment with your particular database environment will determine which filesystem will perform the best.

Optimum Mount Options

- ext4 - `noatime,nodiratime,max_batch time=0,nobarrier,discard`
- XFS - `nobarrier,discard`

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

Tuning Linux

Many Linux variables exist that can be tuned to extract the best performance from the Nytro card. Some of these might perform better than others, but when used as a whole they will benefit in more mixed environments. These variables can be set in a number of different ways; the recommendation is to use the script that is referenced in the next section on how to persist these variables across system reboots.

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

```
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/sdX/queue/iosched/fifo_batch
echo 0 > /sys/block/sdX/queue/iosched/front_merges
echo 5 > /sys/block/sdX/queue/iosched/writes_starved
```

NOTE: These Deadline tuning variables represent a good starting place to test. During your own testing, performance could improve by adjusting these parameters.

For data warehouse or data analytics type of applications/ databases, the following recommendations are best suited for these environments:

```
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 4096 /dev/sdX
echo 0 > /sys/block/sdX/queue/add_random
echo 2 > /sys/block/sdX/queue/rq_affinity
echo 1 > /sys/block/sdX/queue/iosched/fifo_batch
echo 0 > /sys/block/sdX/queue/iosched/front_merges
echo 5 > /sys/block/sdX/queue/iosched/writes_starved
```

Set `swappiness` to 0:

- How to set in a non-persistent value: `sysctl -w vm.swappiness=0`
- How to store in a new persistent value: add `vm.swappiness=0` in the `/etc/sysctl.conf` file.

Invoke 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 common with a MySQL database.

To invoke the JEMALLOC memory allocator instead of using the default memory allocator from glibc, follow these steps:

1. Download and install JEMALLOC for the correct Linux release and version
2. `LD_PRELOAD=/usr/lib64/libjemalloc.so.1` (file location could be different based on OS and release)

3. Add this environment variable to the `/etc/init.d/mysql` script before the statements that execute `mysqld_safe`:

```
case "$mode" in
 'start')
    # Start daemon
    # Safeguard (relative paths, core dumps..)
    cd $basedir
    echo $echo_n "Starting MySQL"
    LD_PRELOAD=/usr/lib64/libjemalloc.so.1
    export LD_PRELOAD
    if test -x $bindir/mysqld_safe
```

4. Restart MySQL:

```
/etc/init.d/mysql restart
```

5. To verify if MySQL is using JEMALLOC, find the MySQL PID then execute:

```
pmap -x MySQL_PID
pmap -x 5736
5736:   /bin/sh /usr/bin/mysqld_safe --
datadir=/u04/datadir --pid-file=/u04/
datadir/MegaraidCL2.pid
```

Invoke Huge Pages

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

To check if the system is setup for HugePages, execute:

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

If any of the values are greater than 0, then the system has been modified to enable HugePages. Now we just have to see if the number of HugePages is large enough for MySQL.

To setup HugePages for MySQL, you need to calculate how much memory MySQL is using, including all of its buffers and memory pools. To calculate the memory allocation that MySQL is taking up, execute the following in MySQL:

```
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 1G+ for the other buffers/pools. I allocated 18GB in HugePages by executing the following commands:

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

Each page is normally 2MB, so a value of 20, for example, will allocate 40MB of memory. This command allocates physical memory, so this much memory must be available. To set the number of pages to allocate, modify `/etc/ sysctl.conf` to add or modify the `vm.nr_hugepages` entry:

```
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 (18GB in this case).

```
echo 18874368000 > /proc/sys/kernel/shmmax
```

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

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

Add "`large-pages`" to the mysqld section of `my.cnf` to enable `HugePages` Modify `/etc/security/limits.conf` to set `memlock` to unlimited for the MySQL user:

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

Add "`ulimit -l unlimited`" to the beginning of the `mysqld_safe` script.

```
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 7864 HugePages are used in this Linux and MySQL environment.

Persist Linux Environment Variables for PCIe-based Devices Across Reboots

In a Linux server, there are times when device assignments change after reboots. Sometimes the Nytro card can be assigned `/dev/sda`. Other times it can be assigned `/dev/ sdd` or another device name with the pattern `/dev/sdX`. This variability could cause a challenge when modifying the Linux environment variables. To avoid this challenge, assignments using the SCSI address should be used so all of the Linux performance variables will persist properly across reboots.

NOTE: If using a filesystem, use the device UUID address in the mount statement in `/etc/fstab` so the mount command will be persisted across reboots.

When the operating system is booted it will assign a name to the Nytro card. For example, the device name can be assigned as `/dev/sdX` where "X" can be any letter. The output from the `ls` command below will show the SCSI address for this Nytro card. To determine the SCSI address of your Nytro card, issue the following command:

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

NOTE: Be sure to make a note of this address, and don't use the address that has '`-partX`' in it.

Now create a script that will be run during start up in `/etc/ rc.local` by copying the code below into a file called "`nwd_getdevice.sh`". Be sure to replace the SCSI address with the SCSI address of the device on your system as determined by the `ls` command above.

NOTE: It is important to include one space between the SCSI address and the closing quote mark.

Contents of `nwd_getdevice.sh`:

```
ls -al /dev/disk/by-id |grep
        'scsi-3600508e07e726177965e06849461a804 '
        |grep /sd      > nwddevice.txt
awk '{split($11,arr,"/"); print arr[3]}'
        nwddevice.txt > nwd1device.txt
        variable1=$(cat nwd1device.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 permission of the file to "execute" and then place this command in the `/etc/rc.local` file:

```
/path/nwd_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.

If you plan to deploy multiple Nytro cards in the server, the easiest way is to duplicate all of commands in the `nwd_getdevice.sh` script and append them to the end. Then edit the SCSI address of the next card and overlay the SCSI address in the newly pasted area. You can follow this procedure for all the Nytro cards installed in the server. An example of this could be:

```
   Contents of nwd_getdevice.sh:
ls -al /dev/disk/by-id |grep
'scsi-3600508e07e726177965e06849461a804 ' | grep
        /sd > nwddevice.txt
awk '{split($11,arr,"/"); print arr[3]}' nwddevice.txt >
nwd1device.txt
variable1=$(cat nwd1device.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
# *********
# second nwd_getdevice.sh
# *********
ls -al /dev/disk/by-id |grep
'scsi-2ndscsiaddr12345666666544444444444 ' | grep
        /sd  > nwddevice.txt
awk '{split($11,arr,"/"); print arr[3]}'
nwddevice.txt > nwd1device.txt
variable1=$(cat nwd1device.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
```

Tuning MySQL

The next logical step in tuning a MySQL database server is applying tuning settings to the MySQL database itself. There are many possible database variables to set to configure a MySQL database. For online transaction program database, for example, the tuning options can be quite different than the settings for data warehouse/analytics types of databases. In performing OLTP type of benchmarks, the following tuning settings have been applied to the MySQL database to get the best performance possible while using the InnoDB database engine.

Parameter	Setting
innodb_log_file_size	4G
innodb_file_per_table	ON
innodb_buffer_pool_instances	8
innodb_io_capacity	20000
default_storage_engine	InnoDB
innodb_flush_method	O_DIRECT
innodb_buffer_pool_size	80% of RAM
innodb_use_native_aio	ON
innodb_read_io_threads	64
innodb_write_io_threads	64
innodb_flush_neighbors	0
innodb_spin_wait_delay	6 (Default OK for small servers, larger value for bigger servers)
innodb_lru_scan_depth	1024
binlog_order_commits	1
key_buffer_size	16m
read_buffer_size	1m
read_rnd_buffer_size	1m
sort_buffer_size	1m
innodb_additional_mem_pool_size	128M
innodb_flush_log_at_trx_commit	1
innodb_log_buffer_size	4M
innodb_log_files_in_group	4
innodb_write_io_threads	64
innodb_read_io_threads	64
performance_schema =	ON
innodb_adaptive_hash_index	OFF

A couple of parameters to look into for further tuning your database include:

Parameter	Setting
innodb_thread_concurrency	>0
innodb_concurrency_tickets	higher for OLAP, lower for OLTP

These parameters address the InnoDB thread scheduler that controls how threads are executed. A good reference for adjusting these parameters for particular environments is in the “InnoDB Concurrency Configuration” section of High Performance MySQL (O’Reilly).

Set Linux variable TMPDIR to a temporary directory for MySQL. 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
```

Recap

Implementing the Nytro card into a MySQL database infrastructure can dramatically increase database performance. By following these simple tuning tips outlined in this paper, a successful implementation of the Nytro Flash Accelerator card for optimal performance can be acheived in most environments.

seagate.com

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