Oracle Linux and MySQL TPC-C Optimizations When Implementing the Sun Flash Accelerator F80 PCIe Card

ORACLE WHITE PAPER | OCTOBER 2014

Table of Contents

Introduction

The Sun Flash Accelerator F80 PCIe Card can help increase the performance of a database application, such as MySQL on an Oracle Linux platform. Once the Sun Flash Accelerator F80 PCIe 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 Sun Flash Accelerator F80 PCIe Card should be configured for optimum performance, but there are tools available and best practices to assist with this effort.

Oracle Linux with Unbreakable Enterprise Kernel (UEK) was chosen for this study because of its upstream support for the latest hardware relevant to modern data center operations. In addition, MySQL database workloads benefit from the platform’s deep integration with the solution stack, optimizations resulting from industry collaborations and enhancements in the UEK.

Oracle Linux with UEK includes extensive performance and scalability improvements to the process scheduler, memory management, file systems, and the networking stack. It was tuned to perform better and faster on leading-edge x86 configurations that feature many CPU cores and large amounts of main memory, and optimized libraries and system calls help to improve performance for MySQL queries. Because of these optimizations and pervasive testing that occurs within Oracle, Oracle Linux is able to address large transaction capacities and scale well as the number of database users or the number of databases increases.

This document will present tuning and processes that can be applied to increase TPC-C performance when using the Sun Flash Accelerator F80 PCIe Card. The performance optimization recommendations that this document will review includes the following:

- Sun Flash Accelerator F80 PCIe Card configuration
- Oracle Linux configuration
- MySQL database configuration

Many of these optimizations were centered on increasing concurrency, decreasing locks, and allowing more physical IO to the Sun Flash Accelerator F80 PCIe Card. The Oracle MySQL 5.6 InnoDB storage engine was used for all the tests referenced below.

Optimizing Performance with the Sun Flash Accelerator F80 PCIe Card

Aligning the Sun Flash Accelerator F80 PCIe Card

The most important step to perform on the Sun Flash Accelerator F80 PCIe Card is to create a partition that is aligned on a specific boundary (such as 4k or 8k) so each read and write to the flash device will require only one physical input/output (IO) operation. If the Sun Flash Accelerator F80 PCIe Card is not partitioned on such a boundary, then reads and writes will span the sector groups which doubles the IO latency for each read or write request.

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 Sun Flash Accelerator F80 PCIe Card

Typically when deploying the Sun Flash Accelerator F80 PCIe Card for database caching (for example, using the Open Source Flashcache developed by Facebook with MySQL), a single-partitioned Sun Flash Accelerator F80 PCIe 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 Sun Flash Accelerator F80 PCIe Cards for database caching, create a Logical Volume Manager (LVM) over all the Sun Flash Accelerator F80 PCIe Cards to simplify administration. The sfdisk command to create a partition for each Sun Flash Accelerator F80 PCIe 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 will be used for write-thru caching, meaning data will be transparent between disk and cache.

If the Sun Flash Accelerator F80 PCIe Card will be used for persisting data, then multiple Sun Flash Accelerator F80 PCIe Cards will be needed to provide fault tolerance. By using two or more Sun Flash Accelerator F80 PCIe Cards to build the RAID array, this concept will eliminate any single point of failure. There are a number of ways to create a RAID over multiple Sun Flash Accelerator F80 PCIe Cards, two of which are:

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

Identifying the Best RAID Level

Oracle coined the term S.A.M.E., which means Stripe And Mirror Everything, in 1999 and popularized the practice that many database administrators (DBA) and storage administrators continue to follow.

To implement S.A.M.E., first determine how the Sun Flash Accelerator F80 PCIe 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 Sun Flash Accelerator F80 PCIe Cards in a RAID array, either using LVM on RAID or creating a RAID array using multiple device administration (`mdadm`), it is important to understand the access pattern of the IO, 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 IO 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 Sun Flash Accelerator F80 PCIe 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, including the official Oracle Linux documentation:

- http://docs.oracle.com/cd/E37670_01/E41138/html/ol_part3_adm.html
- 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 IO 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.

When considering mount options, several options exist that can be applied to increase performance of the Sun Flash Accelerator F80 PCIe Card. For both ext4 and XFS filesystems, the recommendations are:

OPTIMUM MOUNT OPTIONS

Type	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 Oracle Linux

Many Oracle Linux variables exist that can be tuned to extract the best performance from the Sun Flash Accelerator F80 PCIe 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[1]
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
```

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[2]
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
```

1This command should not be necessary when running Oracle Linux with UEK because deadline is the default scheduler in UEK. When running with RHCK include this command to set the scheduler.

2 This command should not be necessary when running Oracle Linux with UEK because `deadline` is the default scheduler in UEK. When running with RHCK include this command to set the scheduler.

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
Address           Kbytes     RSS   Dirty Mode   Mapping
00007faf99a79000     192      96       0 r-x--  libjemalloc.so.1
00007faf99aa9000    2048       0       0 -----  libjemalloc.so.1
00007faf99ca9000       8       8       8 rw---  libjemalloc.so.1
```

Invoke Huge Pages

Instead of using 4k memory pages, Oracle 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 the operating system 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 (18G 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 Oracle Linux and MySQL environment.

Persist Linux Environment Variables for PCIe-based Devices Across Reboots

In an Oracle Linux server, there are times when device assignments change after reboots. Sometimes the Sun Flash Accelerator F80 PCIe 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 Oracle 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 Sun Flash Accelerator F80 PCIe 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 Sun Flash Accelerator F80 PCIe Card. To determine the SCSI address of your Sun Flash Accelerator F80 PCIe 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 (highlighted in yellow) 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
```

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 Sun Flash Accelerator F80 PCIe 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 Sun Flash Accelerator F80 PCIe 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
# *********
# second Sun Flash Accelerator F80 PCIe Card
# *********
```

```
ls -al /dev/disk/by-id |grep 'scsi-2ndscsiaddr123456666665444444444 ' | 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
```

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.

```
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_indexOFF
```

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

```
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 'High Performance MySQL' book by O'Reilly in section 'InnoDB Concurrency Configuration'.

Set Oracle 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
```

Restart MySQL.

Conclusion

Implementing the Sun Flash Accelerator F80 PCIe 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 Sun Flash Accelerator F80 PCIe Card with optimal performance can occur in most environments.

By implementing the Sun Flash Accelerator F80 PCIe Card in an Oracle Linux with UEK and MySQL, there are benefits from Oracle's deep integration of the MySQL database solution stack, as well as optimizations resulting from industry collaborations and enhancements in Oracle Linux with the Unbreakable Enterprise Kernel.

Resources

- **Oracle Sun Flash Accelerator F80 PCIe Card** : http://www.oracle.com/us/products/servers-storage/storage/flash-storage/f80/overview/index.html
- **Oracle MySQL Database** : http://www.oracle.com/mysql
- **Oracle Linux** : http://www.oracle.com/linux

Author: Rick Stehno, Principle Database Engineer, Seagate Technologies
Contributing Author: Phillip Goerl, Oracle, Zeynep Koch, Oracle

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200