SSD Tuning for Linux

About

Notes on optimizing FreeSWITCH™ performance using Solid State Drives and especially preventing early failures of these unique storage devices.

Preface

I spent a months researching **S**olid **S**tate **D**rives before replacing a **H**ard **D**isk **D**rive in July 2012. I learned two things: First, SSDs have unique characteristics that if taken into account, can improve speed and longevity. Second, most people are not aware of the considerations or don't bother with them. This information was compiled from months of research and testing, and is not unique to FreeSWITCH™. However, FreeSWITCH™ can benefit from the much faster startup times. This is a minor update to that information. FreeSWITCH™ users be sure to read FreeSWITCH's core.db I/O bottleneck below.

by Mario G

Introduction

This introduces SSD performance considerations and illustrates how to implement a "tuned" SSD on Linux. *This article is not meant to explain the details of SSD operation.* There are many well written articles providing very detailed information, some referenced below. The objective here is to *pull the information together* to quickly obtain SSD performance and longevity. If you are converting from an HDD to an SSD and want it performing at it's best *mak e sure you prepare a plan first!* These considerations are identical for other operating systems but different commands and utilities are required.

The Problem

SSD performance is largely ignored because it's so easy to do so! When most people replace an HDD with an SSD they simply partition the SSD and copy data to it. If tuning steps are not performed the SSD will still function like an HDD, however, *it is not a good idea*. Unless SSD tuning considerations and preparation steps are taken:

1. The SSD can't provide its best possible performance.
2. Rewrite performance is degraded.
3. SSD performance degrades faster over time. In one benchmark, it got slower than an HDD!
4. The life of the SSD is reduced.

What You Should Know

The Basics

If you just want to get it done without understanding the mechanics jump to the Objectives section. However, if you want to understand *the purpose* of the actions below you must understand that unlike an HDD, an SDD has characteristics that are very important to remember for tuning:

- SSDs cannot overwrite existing data, data must be erased before writing to the same location.
- SSDs are divided into blocks.
- Blocks are divided into pages.
- Data can be written to an empty page.
- Only blocks can be erased.
- In order to reuse pages that contain invalid data, the valid data must be first copied to another block, then the old block erased. This is called "Garbage Collection".

SSD trim, Partition Alignment, and Erase Block Size

SSD functional details are complex and there are many well written articles to explain them, during this project I compiled about 20 very good web sites. The following are a great place to start:

- http://www.anandtech.com/show/2738 <- The SSD Anthology article, one of the best ever written!
- http://en.wikipedia.org/wiki/Write_amplification
- http://en.wikipedia.org/wiki/Flash_memory#Block_erasure
- http://en.wikipedia.org/wiki/TRIM
- http://www.anandtech.com/show/2614/3
- https://wiki.archlinux.org/index.php/SSD#Tips_for_Maximizing_SSD_Performance
- http://wiki.debian.org/SSDoptimization

Objectives

There are two objectives:

- Reduce the *number* of writes, accomplished using tmpfs, trim support, and other techniques.
- Reduce the *size* of data written, accomplished with partition alignment and formatted block size.

Decision Time

Select Your SSD

Make sure the SSD has trim support and is known to work on your operating system, especially if GPT partitioning will be used. All SSD release since 2013 support trim. Note that the trim command can invoke a slowdown in an SSD due to the duration of the function, this is offset by the benefits. The SATA 3.1 specification provides for a new Queued Trim Command, that allows SATA SSDs to execute trim without impacting normal operation. Hopefully, this will eventually show up in SSDs and Linux support soon.

Decide Partitioning Format

Most people will keep the same partition table format as the HDD to be replaced, however, consider changing if it uses the old and ancient **M**aster **B**oot **R**e cord format. The newer **G**UID **P**artition **T**able format has many advantages, some are:

- GPT is required for UEFI motherboards
- Support many more partitions
- Support much larger partitions
- Keeps a duplicate partition table for recovery

The reason to decide now will become obvious later. Consider these issues during your decision:

- Does the motherboard use BIOS or UEFI?
- Does your boot mechanism or boot manager support GPT?
- Will the SSD be used on future UEFI motherboard upgrades?

In our case, the original HDD used MBR partitioning, and LVM circumvented the limits of MBR since there were many partitions. Our SSD uses GPT partitioning for the following reasons:

1. Eliminate the MBR partition limits, although not as elegantly as LVM.
2. Eliminate LVM since there could be potential trim issues, also reduces complexity.
3. All future motherboards will be UEFI and GPT is required for UEFI.

Decide Which File System to Use

Activating SSD trim support is mandatory tuning. Trim is not supported for all file systems on all Linux kernels so it's very important that you select a file system that has trim support in your kernel version. Starting with 3.7 of the kernel, mounting with trim is supported for btrfs, ext4, (v)fat, gfs2, jfs, nilfs2, and xfs file systems. There is an fstrim utility that can be used with btrfs, ext3, etx4, ocfs2, and xfs.

Increase Use of tmpfs

Since reducing writes to the SSD is the major objective you should decide what directories can be moved to tmpfs. I decided to move /tmp, some /var and even thumbnails to tmpfs. Another critical item to move to tmpfs is the web browser and/or server cache. You will have to decide how much you can move based on available memory, the more the better. Of course, keep in mind all tmpfs is cleared with each reboot.

Journalling

Some sites recommend turning journalling off for the file system to reduce writes. I felt more comfortable leaving journalling on which turned out to be a wise decision since there were kernel crashes during this work unrelated to the SSD. No data was lost at any time. If you decide to turn journalling off, a web search will show you how.

AHCI is Required!

SSD trim support is very important! The Linux kernel requires the SATA controller set to AHCI mode to use trim support. Most motherboards have an option in the BIOS or UEFI, just make sure its set for AHCI.

Prepare the SSD for Partitioning

Update Firmware

Always use the latest firmware since SSDs are far more complicated than HDDs and there are a constant stream of fixes from vendors. At the time of this writing the vendor with the GPT problem is still fixing it. Our Intel SSD had a firmware update before the install.

Perform a Secure Erase

Never take an SSD that has been used and simply copy over it! If you read about how SSDs work you learned that previous data can cause a slowdown, and an *operating system format will not resolve the problem, in fact, it may make it worse*. To demonstrate the issue let's compare the SSD and HDD with this process:

1. Take two previously used identical HDDs.
2. Partition and format them with the operating system identically.
3. Copy identical data to both HDDs, or use a disk duplicator.

Both HDDs will have identical physical locations for all of the data. Now perform the same steps on two identical SSDs with one or both of them previously used. Although the OS will report identical sectors, directories, etc. *the true physical location of data will be different* and *they will perform differently*. Any previously used SSD must first be *Security Erased* prior to partitioning to restore the SSD to it's factory state and closer to its factory performance. Here are the steps that worked on this system:

Examine the SSD options with the Linux hdparm command (change sdX to your SSD):

```
hdparm -I /dev/sdX
```

The response should look something like this:

```
Security:
 Master password revision code = 65534
            supported
     not    enabled
     not    locked
     not    frozen
     not    expired: security count
            supported: enhanced erase
     2min for SECURITY ERASE UNIT. 2min for ENHANCED SECURITY ERASE UNIT.
```

If you see "not frozen" proceed with the next step. If you see "frozen" then you cannot proceed. The most common way to unfreeze is to pull the data and power plugs out of the SSD and reinsert them. If this does not solve the issue you will need to research how to unfreeze your model of SSD.

In order to perform a security erase you must first set a password by issuing the command (Abcd is the password):

```
hdparm --user-master u --security-set-pass Abcd /dev/sdX
```

Check the a password is set:

```
hdparm -I /dev/sdX
```

You should see "enabled" under "Master Password". In my case "frozen" was also set on so I unplugged/plugged the SSD as mentioned above to unfreeze.

Perform the SSD security erase:

```
time hdparm --user-master u --security-erase Abcd /dev/sdX
```

This not only resets all the cells but clears the password as well.

Additional information can be found at https://wiki.archlinux.org/index.php/SSD_Memory_Cell_Clearing and many other sites.

Partition the SSD

Partition Alignment

Partition alignment turns out to be one of the most important items to get right but is very complicated. There are many web articles about partition alignment that you can spend weeks reading. I decided to use the gparted live CD to partition because it has an easy option to align partitions on a megabyte boundary which solves the problem. That wastes a few bytes but gets it right. I highly recommend aligning on 1 meg boundaries to avoid the issue. Getting this wrong will cost a lot of time since you have to start over with a security erase.

Partition Table Type

Remember the MBR/GPT decision? Now is the time to use it. If you're not changing from MBR to GPT skip to Formatting the File System, otherwise, determine if a special "bios_boot" partition must be added to the beginning of the SSD:

- If the SSD is to use MBR then the motherboard must be BIOS and no bios partition is needed.
- If the SSD is to use GPT and the motherboard is UEFI then no bios partition is needed.
- If the SSD is to use GPT and the motherboard is BIOS then a special bios_boot partition is required.

This motherboard is BIOS and the SSD is GPT so bios_boot is required as partition number 1. GPARTED was used to create the following partitions:

```
Number  Start   End     Size    File system     Name  Flags
 1      1049kB  3146kB  2097kB                        bios_grub
 2      3146kB  135MB   132MB   ext4
 3      135MB   32.3GB  32.2GB  ext4
 4      32.3GB  64.6GB  32.2GB  ext4
 5      64.6GB  96.8GB  32.2GB  ext4
 6      96.8GB  129GB   32.2GB  ext4
 7      129GB   138GB   8590MB  linux-swap(v1)
```

Grub2 is the boot manager, it requires a 2 megabyte bios partition with the bios_grub flag turned on. After the partitions were created the flag was turned on using the parted command:

```
parted /dev/sdX
   set 1 bios_grub on     <- type this as input to parted
```

When grub2 is installed as the boot manager to this SSD, it will automatically load the bios partition. There is no need to ever reference it. If a different boot manager is used you will need to find the GPT requirements for your boot manager although most are very similar.

Formatting the File System

Unlike an HDD, when data is rewritten to an SSD the old data is deleted and the new data written to a different area, the old data must eventually be erased by the SSD. Another difference is that although the smallest data that can be written is a page such as 4k, the smallest data that can be erased is the much larger erase block size which may be 512k! If your SSD is MLC then 4k formatting will be fine, if it's an SLC the erase block size is much smaller and varies depending on SSD model. The details are described in the links provided above. Most often the default file system block size is fine, although you may want to look into this if your SSD uses SLC flash.

Prepare FSTAB Entries

You should prepare the new FSTAB entries *before* copying a source drive to avoid copying data moved to tmpfs. Changes made to this system include:

- Added discard to the ext4 file system to activate SSD trim support.
- Added noatime to reduce writes, note that it is known some applications have problems with this but they are rare.
- Added nodiratime to reduce writes.
- Moved some directories to tmpfs.
- Created /ramdisk for Firefox cache but could have used Firefox config to replace disk cache with memory cache.

This is the FSTAB used, the first six are the SSD partitions:

```
LABEL=SSDsuse121     /                    ext4 acl,noatime,nodiratime,discard   1 1
LABEL=SSDsuse121m    /suse121m            ext4 acl,noatime,nodiratime,discard   0 2
LABEL=SSDcommon      /common              ext4 acl,noatime,nodiratime,discard   0 2
LABEL=SSDsuse122m    /suse122m            ext4 acl,noatime,nodiratime,discard   0 2
LABEL=SSDgrub        /grub                ext4 noacl,noatime,nodiratime,discard 0 2
LABEL=SSDswap        swap                 swap defaults              0 0
proc                 /proc                proc defaults              0 0
sysfs                /sys                 sysfs noauto               0 0
debugfs              /sys/kernel/debug    debugfs noauto             0 0
usbfs                /proc/bus/usb        usbfs noauto               0 0
devpts               /dev/pts             devpts mode=0620,gid=5     0 0
tmpfs                /ramdisk             tmpfs nodev,nosuid,noatime,mode=1777,size=100M 0 0
tmpfs                /tmp                 tmpfs defaults,noatime,mode=1777   0  0
tmpfs                /var/spool           tmpfs defaults,noatime,mode=1777   0  0
tmpfs                /var/tmp             tmpfs defaults,noatime,mode=1777   0  0
#  .thumbnails files are never deleted and can grow quite large over time.
tmpfs                /root/.thumbnails    tmpfs defaults,noatime,mode=1777   0  0
tmpfs                /home/mario/.thumbnails    tmpfs defaults,noatime,mode=1777   0  0
```

Prepare the Source Drive

Before copying data from the source drive you should delete all of the files you decided to move to tmpfs. Once the data is copied and the new fstab is used you will not have access to the old data. It's best not to copy it in the first place anyway. If you are booted from the partition you are to copy from it would be better to copy it somewhere else first so the files can be deleted easily. If you turned off browser disk caching, etc. don't forget to delete the old HDD caches before copying to the SSD.

Linux Tuning

Elevator Setting

Most Linux systems use I/O scheduling algorithms designed for HDDs. For instance, some order disk writes to the same device to reduce total seek time. Since SSDs are so fast and don't have any seek time, the I/O scheduler for an SSD should be deactivated because it adds unnecessary overhead. There are two ways to achieve this:

1. If the only drive(s) in the system are SSD, add elevator=noop to the kernel boot options. This is typically done in the boot manager configuration.
2. If the main drive is an SSD and there is a very low use HDD, consider adding elevator=noop to the kernel boot options.
3. if the system uses SSDs and HDDs heavily, noop should be activated only for the SSDs, not the HDDs. This usually involves adding a startup script. How this is accomplished varies depending on the Linux distribution so you need to do some basic research for your distribution. You may find this useful: https://wiki.archlinux.org/index.php/SSD#I.2FO_Scheduler

Swappiness

This only applies if the swap file is moved to the SSD. The swappiness value affects the activity of the swap file. If you placed the swap file on the SSD you should *reduce* the swappiness of Linux. There are many good explanations but simply put: Linux tries to anticipate when something *might* need to be swapped and does it anyway. By reducing swappiness Linux is less likely to do this so swap writes are reduced. Swappiness is a value from 0 to 100, the higher the more swappiness. You can check your current value using:

```
cat /proc/sys/vm/swappiness
```

There are a couple of different ways to change swappiness permanently. I changed this system from it's default of 60 to 1 by adding the following to /etc /sysctl.conf:

```
vm.swappiness=1
vm.vfs_cache_pressure=50
```

Research showed that reducing swappiness when using HDDs also resulted in better performance.

Never Do This!

- Never perform defragmentation, it serves no purpose but to increase SSD wear and slow it down.
- Never use utilities that overwrite a file repeatedly such as shred, unlike an HDD each SSD rewrite is to a different area, the rm command is sufficient.

FreeSWITCH's core.db I/O bottleneck

On a normal configuration, core.db is written to disk almost every second, generating hundreds of block-writes per second. To avoid this problem, turn /usr /local/freeswitch/db into an in-memory filesystem. If you use SSDs, it is CRITICAL that you move coe.db to a RAM disk to prolong the life of the SSD.

On current FreeSWITCH versions you should use the documented "core-db-name" parameter in switch.conf.xml (simply restart FreeSwitch to apply the changes):

```
<param name="core-db-name" value="/dev/shm/core.db" />
```

Otherwise you may create a dedicated in-memory filesystem, for example by adding the following to the end of /etc/fstab

```
#
# Example of /etc/fstab entry (using default size)
#
tmpfs /usr/local/freeswitch/db tmpfs defaults 0 0
#
# To specify a size for the filesystem use the appropriate mount(1) option:
#
# tmpfs /usr/local/freeswitch/db tmpfs defaults,size=4g 0 0
#
```

To use the new filesystem run the following commands (or the equivalent commands for your OS):

```
mount /usr/local/freeswitch/db
/etc/init.d/freeswitch restart
```

An alternative is to move the core DB into an ODBC database, which will move this processing to a DBMS which is capable of handling large numbers of requests far better and can even move this processing onto another server.