Using a Solid State Drive with FreeBSD

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
	2016-01-10		WB

Contents

Last updated 2016-01-10

Available in HTML or PDF. Links to all my articles here. Created with AsciiDoc.

Solid State Drives have become practical hard drive replacements in many cases. There are important differences from hard drives, and careful setup will pay off in long-term performance.

These instructions are written for FreeBSD 10.X and later.

1 Terms

AHCI	*Advanced Host Controller Interface* is a standard for disk drives and controllers. It offers advanced features, but for our purposes the important part is that enabling AHCI in the BIOS usually gives a 5-15% performance increase. Not all controllers offer AHCI; check your chipset and BIOS.
TRIM	Most SSDs support a feature called TRIM. TRIM is just a status update: when the filesystem deletes a block, it also notifies the device that the block is no longer in use. The SSD firmware can then do its behind the scenes wear leveling. Without TRIM, it would not be able to tell whether that block was still in use. TRIM does not require AHCI.

2 Warnings

Do not enable TRIM unless the SSD supports it. Better yet, avoid SSDs that do not support TRIM.

Enabling AHCI in the BIOS after installing an operating system could change the way that operating system or others see the attached drives. This is a particular problem with FreeBSD installed in a multi-boot system. Enable AHCI before installing the operating systems, or disable it temporarily to update drivers on the ones that do not see the disk when AHCI is enabled.

The version of *gpart(8)* that came with FreeBSD 9.0 did not do what was expected when both *-a* and *-b* options were used. That was fixed by FreeBSD 9.1, but to remain compatible with 9.0 the commands shown here use one or the other.

The features used here have been tested, but not with every possible configuration of FreeBSD. Enabling crash dumps but having swap in a file is probably a mistake. MBR partitioning may be required when multiple operating systems boot from the same drive. Consider these suggestions, and adjust to fit your usage.

3 Quick Summary

For easy reference, the partitioning and formatting steps explained below are shown in a single lump here.

```
# gpart create -s gpt ada0
# gpart add -t freebsd-boot -s 512k -a4k -l ssdboot ada0
# gpart bootcode -b /boot/pmbr -p /boot/gptboot -i1 ada0
# gpart add -t freebsd-ufs -l ssdrootfs -b 1m -s 2g ada0
# gpart add -t freebsd-ufs -l ssdvarfs -a 1m -s 2g ada0
# gpart add -t freebsd-ufs -l ssdusrfs -a 1m ada0
# newfs -U -t /dev/gpt/ssdrootfs
# newfs -U -t /dev/gpt/ssdvarfs
# newfs -U -t /dev/gpt/ssdusrfs
```

4 Partitioning an SSD

First steps: create a GPT partition scheme, a partition to hold the FreeBSD loader, and install the bootcode. The boot partition is 512K, the largest safe size due to code limitations in the loader. While the boot code is currently much smaller than 512K, there is no real reason to make the partition smaller. And if the boot code gets larger in the future, as it has before, the partitions will not need to be rearranged.

The boot partition here is 4K-aligned, but it really does not matter. A boot partition is only read or written rarely, and the code is so small that alignment is not necessary. Here it is aligned anyway, just for consistency.

```
# gpart create -s gpt ada0
# gpart add -t freebsd-boot -s 512k -a4k -l ssdboot ada0
# gpart bootcode -b /boot/pmbr -p /boot/gptboot -i1 ada0
```

Now it gets more interesting. In my experience, a 2G root partition is large enough to hold a traditional configuration and allows space for the current kernel and the *kernel.old* directory created when a new kernel is built.

GPT labels are assigned to each GPT partition to make the drive easy to relocate to a different controller or computer. Please try to keep your drive labels unique. Identical labels on different drives could lead to a surprising and probably disappointing result.

```
# gpart add -t freebsd-ufs -l ssdrootfs -b 1m -s 2g ada0
```

Note that the partition starts at 1M. Why do that? The 1M location is 4K aligned. It is also 128K and 1M aligned, which can be important for some SSDs. Ever since Windows Vista, Microsoft has started their data partitions at 1M. And some RAID controllers also use a 1M or smaller stripe size. Using this location makes non-FreeBSD GPT disk partitioning more likely to work, or at least less likely to destroy data that is located in what the controller sees as a metadata area.

There really aren't any disadvantages. 492K (not meg, K!) wasted, in exchange for complying with a de facto standard.

The starting point and the size of this partition are already aligned values. So it is not necessary to use *-a* here, and leaving it out avoids the problem with earlier versions of *gpart(8)* mentioned above.

Following the root partition is a 2G partition for */var* and the rest of the SSD for */usr*. Actually, a 1G */var* is often large enough. These are aligned to 1M increments, again an even multiple of the 4K block size. Salt to taste.

```
# gpart add -t freebsd-ufs -l ssdrootfs -b 1m -s 2g ada0
# gpart add -t freebsd-ufs -l ssdvarfs -a 1m -s 2g ada0
# gpart add -t freebsd-ufs -l ssdusrfs -a 1m ada0
```

Some things appear to be missing here. There are no partitions for swap or */tmp*. We will set these up in a bit, taking advantage of SSD features.

5 Filesystems and TRIM

FreeBSD's UFS filesystem supports TRIM. That will be enabled with *-t* while formatting the filesystems.

Soft updates are also enabled. Soft updates journaling (SUJ) is **not** used for two reasons: there have been problems with SUJ that prevent the use of *dump(8)* to back up filesystems, and SUJ's killer feature is dramatically reduced *fsck(8)* times. But SSDs provide dramatically reduced *fsck(8)* times anyway.

```
# newfs -U -t /dev/gpt/ssdrootfs
# newfs -U -t /dev/gpt/ssdvarfs
# newfs -U -t /dev/gpt/ssdusrfs
```

6 Restoring Data

Warning
If you install a fresh version of FreeBSD to the SSD, it may *newfs(8)* the filesystems without TRIM. Boot into single user mode and use `tunefs(8) -t enable` to re-enable it on each filesystem.

Warning
Do not use *dd(1)* or other programs that directly write blocks on the SSD. It copies every block, whether they are used or not. From the SSD's point of view, every block is in use, and wear leveling becomes slow or impossible. Then the SSD becomes slow.

The procedure shown here copies data from the existing system drive onto the SSD.

```
# mount /dev/gpt/ssdusrfs /mnt
# dump -C16 -b64 -0aL -f - /usr | (cd /mnt && restore -rf -)
# umount /mnt
# mount /dev/gpt/ssdvarfs /mnt
# dump -C16 -b64 -0aL -f - /var | (cd /mnt && restore -rf -)
# umount /mnt
# mount /dev/gpt/ssdrootfs /mnt
# dump -C16 -b64 -0aL -f - /    | (cd /mnt && restore -rf -)
# umount /mnt
```

7 Fixing */etc/fstab* and */tmp*

/etc/fstab must be changed to refer to the SSD. At the same time, the missing */tmp* will be added.

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/gpt/ssdrootfs      /               ufs     rw              1       1
/dev/gpt/ssdvarfs       /var            ufs     rw              2       2
tmpfs                   /tmp            tmpfs   rw,mode=01777   0       0
/dev/gpt/ssdusrfs       /usr            ufs     rw              2       2
```

tmpfs(5) is used for */tmp*. It uses the virtual memory system to make an efficient memory filesystem. If the system runs low on memory, it will be moved to swap space. This example does not limit the amount of space available in */tmp*, but *tmpfs(5)* gives details on doing that.

This is truly a temporary filesystem. It will be cleared on restart.

8 Swap

Swap space is rarely used but nice to have. Allocating a partition ties up that swap space permanently, and data written to swap will not use TRIM. So we will use a swap file. Because the data goes through the file system, TRIM will be used, and the swap file can be resized without repartitioning the SSD.

Create the swap file with *dd(1)*. This example creates a 4G swap file. 2G is often enough and may be more appropriate for a small SSD.

Warning
It is tempting to use a sparse file here. Congratulations on thinking of it, but that way lies madness. Do not use a sparse file for a swap file.

```
# mkdir /usr/swap
# dd if=/dev/zero of=/usr/swap/swap bs=128k count=32768
```

Edit */etc/fstab* to use the swap file. This uses the new support for swap files added in FreeBSD 10 and described in *fstab(5)*.

```
# Device          Mountpoint       FStype  Options                         Dump    Pass#
md99              none             swap    sw,file=/usr/swap/swap,late     0       0
```

9 Using the SSD

Make sure the SSD is set as the first boot drive. Boot into the system and test it.

Done!