ECE590 Enterprise Storage Architecture Lab #1: Drives and RAID

Now we'll play with those hard drives you have and learn ways to establish redundancy for them.

NOTE: This assignment assumes familiarity with basic UNIX concepts such as IO redirection and piping; if you need to brush up on this content, you can use this Duke UNIX online course.

Directions:

- This assignment will be completed in your groups. However, **every member of the group must be fully aware of every part of the assignment**. Further, while you can discuss concepts with other groups, actual steps and answers should not be shared between groups.
- The assignment will ask for short answers or screenshots; this material should be collected in a file called `ece590-group<NUM>-lab1.pdf`, where <NUM> is your group number (sans brackets!), and submitted via Sakai. *Word documents will not be accepted*. Anything you need to include in this document is highlighted in cyan.

1 Check hard drives for existing data

In Lab 1, you created dummy virtual devices for each physical device in the hardware RAID controller, allowing the OS to see the SSD and each HDD.

The SSDs you have are either new or lightly used last year, but the HDDs are used stock purchased on ebay (and many were used last year, too). Before we overwrite these disks, let's read the entirety of their content into a file on a separate machine. Later in the course, we'll learn about data recovery and forensics, and hopefully at least one of these drives will have something interesting to look at.

We're going to be **imaging** the drive, meaning that we're going to copy each block sequentially into a file known as a **disk image**. Imaging is useful because it occurs *beneath* the filesystem, so you capture all data and metadata without missing anything. This is easy on Linux, because like all UNIX-based operating systems, devices are represented as files, so the process of disk imaging is equivalent to a simple file copy.

This is also a chance to practice our UNIX command-line skills, as with a little cleverness, we can make this disk imaging very efficient. Here are some useful tools:

- **`dd`**: This is a very old tool used to get fine control over sequential reading/writing, like a fancy version of `cat` optimized for block IO. Manpage, examples.
- **`gzip`/`bzip2`**: These are stream-oriented compression tools, meaning they take a sequential stream of bytes and compress/decompress inline. This makes them useful to use in a pipeline.

Gzip is the older tool and offers modest compression at very low CPU usage; bzip2 grants modest compression improvements but is significantly slower. We'll just use gzip. Manpage.

- **`ssh`**: You've no doubt been using SSH to connect to UNIX-style systems, but it can do much more. In this case, it can accept a stream via pipe and hand the data to a command executed on a remote machine. Manpage, examples.
- **`pv`**: This tool works like cat, but it puts up a console progress bar as it reads through the file. It's not a core UNIX tool like the above, so you'll need to install it ("`sudo apt install pv`" on Ubuntu Linux). We can use it to monitor the status of our imaging operation as it goes. Manpage.
- **`lsblk`**: This tool lists the block devices on a system. We can use it to identify our drives. Manpage.
- **`mount`**: A fundamental UNIX tool that attaches and detaches filesystems. Filesystems may be "real" (such as the SSD-based ext4 filesystem that is your root directory) or "virtual" (such as the 'proc' filesystem mounted at /proc which provides basic info from the kernel about the running system). Manpage.

A fortunate side-effect of imaging these drives is that this will also serve as a basic health check for them. I fully expect one or more to contain *bad sectors*, which will be reported as read errors during the imaging. If you encounter read errors during imaging:

- Document the errors in your write-up, including which drive(s) it was, messages received, etc.
- Contact the instructor to swap drives with a spare; document the replacement in your write-up.
- Run "`dmesg`" to view the kernel log, and include any entries that appear to deal with hardware-level errors on the drive in question in your write-up (see here for an example of kernel log messages relating to drive hardware errors).

Now let's get started.

1.1 Examine block devices

Using the lsblk command, take a look at your drives. Screenshot.

In Linux, drives are represented by /dev/sda, /dev/sdb, etc. You should see your SSD (distinguished by its size of around 120GB) plus your HDDs (each around 73 or 146GB depending on your model). Partitions that have been made are indicated with numbers, so /dev/sda1 is the first partition of the first drive. Based on where you installed your SSD (in the first drive slot) and how you partitioned it, I'd expect /dev/sda1 to be the partition that contains your root directory (though it's okay if it's not).

Let's see what's mounted: simply run 'mount' without arguments. Screenshot.

You should see something like "`/dev/sda1 on / type ext4 (rw)`", which indicates that your root filesystem is on /dev/sda1. Make note of which drive your OS is on (`/dev/sda` in this example), as we won't want to image that. Note your OS drive in the write-up.

All other /dev/sdX drives will be imaged.

1.2 Image the drives

Let's construct a pipeline that will (1) read the drive while reporting status as we go, (2) compress the stream, (3) send it to another server. The destination server will be "storemaster", the reference server configured by the instructor. An account has been made on this server to let each group store image files, and sufficient storage is available to hold images of each drive we have. The username you will use is simply "image", and the instructor will provide the password.

To image /dev/sdb, type:

```
sudo pv /dev/sdb | gzip | ssh image@storemaster.egr.duke.edu dd of=/image/esa00-sdb-20170822.img.gz
```

In your write-up, explain what each part of this command does and why.

To keep each group's images separate and well labeled, use a filename of the format:

`/image/esa<NUM>-<DRIVE>-<DATE>.img.gz`

Where <NUM> is your group number, <DRIVE> is the device (sda, sdb, etc.), and <DATE> is the current date, in YYYYMMDD format.

Go ahead and image all non-OS drives on your system.

The compression process will be the performance bottleneck, but because you have multiple cores, you can go ahead and image the drives in parallel. This process will take a while, so if getting disconnected from SSH is an issue, consider learning to use screen to allow your process to run in the background.

1.3 Examine the drives in more detail

Let's glance at once of these drives. We'll get into data forensics later, but you're probably curious if there's anything here. Our main question should be "were these drives *zeroed* before being sold?". To "zero" means to overwrite all bytes with zeroes, and is the standard method of cleaning a drive. We can quickly check by running "hd" (hex dump) on the drive. If non-zero bytes come up, there's probably something there to find. If it just prints a row of zeroes and seems to wait, it's indicating that all subsequent bytes so far are also zeroes, and we likely have a blank drive.

Quickly check each drive with: `sudo hd /dev/sda`

Example of a zeroed drive:

```
$ sudo hd /dev/sdd
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

(It sits there reading, so I get bored and press Ctrl+C after a few seconds)

Example of a drive with stuff on it:

```
$ sudo hd /dev/sdb
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001b0  00 00 00 00 00 00 00 00  f1 c4 1f 21 00 00 00 20  |...........!... |
000001c0  21 00 83 44 48 05 00 08  00 00 89 03 40 00 00 00  |!..DH.......@...|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 40 e8 00 00 00 a1 03  33 73 2e 00 67 b1 91 03  |.@......3s..g...|
```

(Stuff flies by real fast, so press Ctrl+C to stop it)

Note which drives contain data and which appear to be zeroed in your write-up.

2 HDD vs. SSD performance

2.1 Benchmarking

Let's do a very simple performance test on our drives. We want to read each device and determine the IOs Per Second (IOPS). The IO in this case will be block-oriented: we'll be reading a chunk of data at a time (e.g. 512 or 4096 bytes). This is because IO operations have a fairly high setup cost (including kernel interrupt and scheduling of the actual IO traffic), so we want to amortize that cost over a significant amount of data.

The benchmark software we'll be using is [the simple "iops" tool written by Benjamin Schweizer]. Download this tool onto your server. It is just a Python script, so just enable it for execution (`chmod +x`) and see the usage message by running it without arguments.

We're going to run it directly on our bare drives (bypassing the filesystem layer), so you'll need to run it as root. It only does read operations, so the test can be done safely even on your OS drive.

Run it in both sequential and random mode for each drive you have (`/dev/sd?`), and run it with block size 4kB and 1MB. Keep the number of parallel threads at 1. Interested readers can use scripting to automate these tests.

In your writeup, include your results as a table, showing the test inputs, the IOPS, and the throughput (MB/s).

Based on these numbers, how long would it take to read your first HDD entirely in sequential order? How long if you did it in random order? How does this make you feel?

Do the same computation for your SSD. Wow, SSDs sure are neat, huh?

Observe that sequential IO is still faster than random IO on the SSD, despite it having no mechanical parts. Why is this?

How did increasing the block size change things? Why might that be?

2.2 Comparing to the specs

Look up the model of HDDs and SSDs you're using. To do this, you'll need to shut down your server and eject the drives to check the labels[1]. Try to look up published specs or benchmarks for your drives. If you can't find the exact drives, look for specs for a comparable drive (e.g. another 73GB 10kRPM SAS drive or another 120GB SATA SSD).

In your write-up, share links to the specs you find, and summarize the performance stats (random IOPS and sequential throughput for each drive type).

Compare the rated random IOPS and sequential throughput to your measurements from the previous question. I suspect the rated ones will be higher (as in, you measured less performance than the manufacturer says you can expect). In this case, how might the manufacturer test differ from ours? In the unlikely case that you actually measured higher performance than the rated specs, why might this be?

[1] Normally, you'd be able to use a tool called hdparm to read the make/model of your drive, but our drives are being "hidden" by the hardware RAID controller as virtual drives.

3 Software RAID

RAID can be done either in software by the operating system or in hardware by a dedicated RAID card. In Lab 1, we configured the hardware RAID card to simply pass each drive through directly to the OS. This will allow us to apply software RAID. In Linux, software RAID is called handled by the "md" subsystem. Ensure the "mdadm" tool is installed.

NOTE: I wrote some of the documentation below without access to a machine with real drives I could trash, so I faked some of the sample output using simulated drives; apologies if my forgeries are imperfect.

3.1 Set up software RAID

Noting this RAID setup guide, create a three-disk RAID5 called /dev/md0 using your HDDs; one drive will serve as a spare device. Document the commands you used to set it up in the write-up.

You can always check the current status of all md RAID devices by reading /proc/mdstat. When I built mine, I checked this file immediately after to see it "repairing" the new array (calculating and storing parity). This may take some time, but the array is usable while it occurs, albeit at a lower performance level. Therefore, **wait for it to finish before proceeding with benchmarking**. Below, you can see mdstat showing the rebuild underway, and then later showing the array as ready:

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sda[4] sdd[3](S) sdc[1] sdb[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2]
[UU_]
      [================>....]  recovery = 80.5% (82836/101888)
finish=0.0min speed=27612K/sec

unused devices: <none>
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sda[4] sdd[3](S) sdc[1] sdb[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]

unused devices: <none>
```

Note the "(S)" flag on sdb: this indicates that this device is a hot spare.

Include similar output of /proc/mdstat either during rebuild or afterward in your write-up.

3.2 Benchmark

Based on the theory behind RAID, compute the logical size your RAID array should be, show your work and note the result in your write-up. Then, using lsblk, fdisk, or another block device tool, confirm the size of the RAID device /dev/md0 on the system, showing the output in your write-up.

Using the exact same procedure as in section 2.1, benchmark the RAID drive /dev/md0.

In your write-up, include your results as a table, showing the test settings, the IOPS, and the throughput (MB/s).

In your write-up, compare all the performance numbers to the average performance of a single one of the drives. Does the performance improvement match the theory?

3.3 Set up a filesystem

Our RAID array is represented by /dev/md0. Now we need a file system on top of it. There are many file systems available, and even a basic install of Linux supports many of them. We'll learn more about the design of file systems later in the course, but for now, we just need to create one so that a directory structure can be accessed on our RAID array for testing. To create the initial filesystem, we use the mkfs ("make file system") utility, which has variants for each supported filesystem (e.g. ext2, ext3, ntfs, etc.). Let's just use the common Linux choice of ext4. **Use "mkfs.ext4" to prepare a filesystem on /dev/md0;** provide a screenshot of this.

Now we need a *mount point*, a directory into which we can attach the filesystem. **Using mkdir, create a directory called "/x".**

Now use the mount utility to mount our /dev/md0 filesystem to /x. Note that while the mount utility does support a lot of options, none are needed for this basic operation – the tool will have no trouble auto-detecting the type of filesystem present and using reasonable default settings when mounting it. When done, you can verify by typing "mount" by itself to list mounted filesystems; you can also type "df" to check free space. Example:

```
root@xub1404dt:~ # mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
 ...
none on /sys/fs/pstore type pstore (rw)
systemd on /sys/fs/cgroup/systemd type cgroup
(rw,noexec,nosuid,nodev,none,name=systemd)
/dev/md0 on /x type ext4 (rw)

root@xub1404dt:~ # df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       19478204 4191128  14274596  23% /
none                   4       0         4   0% /sys/fs/cgroup
udev              489424       4    489420   1% /dev
tmpfs             100020    1456     98564   2% /run
none                5120       0      5120   0% /run/lock
none              500096     152    499944   1% /run/shm
none              102400      40    102360   1% /run/user
/dev/md0          203772    2504    201268   2% /x
```

Provide screenshots or a terminal log similar to the above in your write-up.

3.4 Use the file system

By whatever means you wish, populate the filesystem at least a few kilobytes of content. At least one file should be a text file you can verify the contents of. When done, you should be able to `cd` to your filesystem and see your stuff, e.g.:

```
root@xub1404dt:/x # ls -l
total 781
-rwxrwxrwx 1 root root 797772 Sep 30 17:14 kern.log
-rwxrwxrwx 1 root root     21 Sep 30 17:15 test.txt
root@xub1404dt:/x # cat test.txt
This is my text file
```

Provide screenshots or a terminal log of these steps in your write-up. Be sure to include a file listing and "cat" of your small text file, as shown above.

3.5 Damage the RAID

We'll now simulate failure of one of the disks. As soon as you do, the array will begin repairing itself using the hot spare we provided.

Use the mdadm command to "fail" one of the active drives in the array (not the spare!), then print the content of /proc/mdstat so you see the array status while it's rebuilding. Check the /proc/mdstat file again after it's done to see the final status. You'll find the failed disk with a "(F)" flag, and the former spare disk is now in the array, so it no longer has the "(S)" flag. Example output of mdstat before, during, and after rebuilding:

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd[3](S) sdb[5] sdc[4] sda[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]
```

(failure is simulated)

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd[3] sdb[5](F) sdc[4] sda[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2]
[U_U]
      [==========>.........]  recovery = 50.0% (51580/101888)
finish=0.0min speed=51580K/sec

unused devices: <none>
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd[3] sdb[5](F) sdc[4] sda[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]

unused devices: <none>
```

During reconstruction, verify that the content in the mounted filesystem /x is undamaged, despite a disk failure, e.g.:

```
root@xub1404dt:/x # ls -l
total 781
-rwxrwxrwx 1 root root 797772 Sep 30 17:14 kern.log
-rwxrwxrwx 1 root root     21 Sep 30 17:15 test.txt
root@xub1404dt:/x # cat test.txt
This is my text file
```

Provide screenshots or a terminal log of these steps in your write-up, including evidence of your text file surviving the failure.

3.6 Replace the faulty drive

At this point, because the spare was there to allow rapid reconstruction, your array can again withstand a drive failure without data loss, but there would be no more spares to rebuild it immediately in that case. Running with no spares is dangerous, so we'll now simulate replacing the faulty drive. Steps:

1. Use the mdadm "remove" option to delete the faulty drive /dev/sdb from of the array listing. This is the software equivalent of pulling the drive out of the server.
2. At this point you'd replace the drive with a new one, but since our drive isn't actually bad, just imagine doing so in your mind.
3. Use the mdadm "add" option to add our 'replacement' drive (the same drive) to our /dev/md0 array.

When you're done, replaced drive should show as a new spare device in /proc/mdstat:

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdb[5](S) sdd[3] sdc[4] sda[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]

unused devices: <none>
```

Your array is now capable of immediately rebuilding if another drive failure occurs.

Provide screenshots or a terminal log of these steps in your write-up.

3.7 Tear down software RAID

Use mdadm to destroy and remove your RAID device.

4 Hardware RAID

Let's compare the setup and performance of software RAID to hardware RAID.

4.1 Set up hardware RAID

Either in-person or via remote access, reboot the server, and press the appropriate key to get into the RAID controller setup interface (like you did in Lab 1). Destroy the virtual devices for the hard disks, *taking extreme care not to harm the virtual device for the SSD*, which is where your OS lives.

Then create a new virtual device that is a three-disk RAID5 with a single hot spare (the same as configuration as our previous software RAID).

Take a screenshot of the firmware interface showing the RAID5 setup for your write-up.

4.2 Benchmark

Your new hardware RAID device will show up as a new /dev/sd? device. Figure out which device it is and note the drive size.

Wait for the array rebuild to complete (I believe you can check this in the boot-time menu).

Using the exact same procedure as in section 2.1, benchmark the RAID drive.

In your write-up, include your results as a table, showing the test settings, the IOPS, and the throughput (MB/s).

In your write-up, compare all the performance numbers to the software RAID. Is one strictly better than the other? Are there tradeoffs? What's the difference?

4.3 Tear down hardware RAID and restore single-drive access

Reboot, go back into the RAID controller firmware, destroy the RAID5 virtual device, and in a procedure similar to Lab 1, make pass-through virtual devices for each HDD, basically restoring the system to its previous configuration from before you started this assignment. Again, *take extreme care not to harm the virtual device for the SSD*, which is where your OS lives.