Storage Server

In 2019, we decided to replace the iSCSI server we had built, and which has been running continuously for over five years with no problem. While the existing servers are running fairly well, a concern about the hardware's age, some slowdown on access as we have added more and more consumers of the services, and the possibility of improving on our existing configuration with newer technology and our additional experience.

A storage server can mean many things to many people, but in our case, we were looking for a single machine that would provide an iSCSI target and and NFS service. The consumers of these services are around two dozen Xen virtual machines.

We are a smaller company, so we look for less expensive ways to perform a given task. In this case, we would like to set up a system to handle our projected needs over the next 5 years, but with the ability to adjust if our projections turn out to be too conservative.

Hardware

We have purchased refurbished enterprise grade servers from MET (https://www.theserverstore.com) for over a decade. In all cases, our servers have been reliable for 5-10 years. Because of this, we decided to purchase two HP Proliant DL380P G8's with 25 SAS bays in a 2U case. Additionally,

- We requested they add an HBA controller so we could manage our disks via software raid (mdadm) or zfs.
- We requested one “enterprise grade” ssd for the boot volume and used the motherboard's built in Compact SATA connector to mount it internally.
- Requested 32G of RAM. This came in two 16G DIMM's, and it is probably not enough, so we intend to upgrade to 64G in the near future.
- We decided to try high end consumer grade SSD's for this implementation. The 500G Samsung EVO has good specifications, so we purchased 10 for each machine from the local MicroCenter at about $80/ea.

Total system price was about $1500 per machine.

Operating System

The first decision was which operating system to use. The original system was created using Debian 7 (Wheezy), which has rarely been updated over its lifespan. At the time of this machines creation, Wheezy was the “latest and greatest” of the Debian line, and it has proven bullet proof during the lifespan of these machines.

Due to Debian's increasing dependence on systemD, it was decided to explore alternatives. systemD appears to be leading some Linux installations towards convenience instead of the reliability that it has been so well known for.

Devuan (https://devuan.org/) and others have attempted to reverse the trend by offering (in this case)

a version of the operating system which offers other init systems besides systemD. In our case, we have used Devuan to build reliable servers and workstations using the original SysV init system. Though showing its age, it is rock solid in getting a system up.

The other alternative is a very old version of Unix; BSD. We chose FreeBSD (https://www.freebsd.org/) based on some limited experience with it at our shop. All of our experiences have shown FreeBSD to be rock solid, with convenient upgrade paths even across major releases.

Both Devuan and FreeBSD have great support for iSCSI targets and NFS servers and have proven very reliable.

Devuan is, arguably, a start-up since it has come into existence after the forced implementation of systemD on Debian (2015), but the maintainers have long term experience from the original Debian days.

FreeBSD 1 was released in November 1993, and has grown steadily to the major distribution it is.

File Systems

The original servers used mdadm+lvm2, with drbd to synchronize the “disks” between the two. The disks (7200rpm 1T drives) were configured into a software RAID-6, which became the physical volume for a volume group. Logical volumes were created to handle the operating system, and an additional large logical volume was created on both machines, and set up to be the target of the drbd synchronization. The logical volume was then set as a physical volume for an additional volume group, from which the logical volumes for each image (plus one for nfs) were created. pacemaker was supposed to be configured for automatic failover, but never implemented.

We could replicate this system, which has worked well, but zfs needed to be considered also. zfs is an advanced file system with multiple configurations *on a per directory basis* which makes it very difficult to lose any data, even under adverse conditions. zfs can also replicate most of the functionality mdadm and lvm2 provided on the original system. When configured with hast (same as drbd), we can take the old systems' mdadm+lvm2+drbd+ext4 and use zfs+hast instead.

The main issue is the word “most” above. OpenZFS is not as flexible as mdadm when it comes to dynamically modifying the low level structure of a system. mdadm allows you to build a 4 disk RAID-5, add an additional physical drive and either

- convert to RAID-6
- increase the physical size of the RAID-5 set (ie, convert from 4x1T = 3T space to 5x1T = 4T space)

OpenZFS requires you to export all of your data, destroy the RAIDZ, build a new RAIDZ (or RAIDZ2), then import your data. This requires downtime, and a place to put your data while you make the conversion, all of which is not required with mdadm (all work can be done while the system is running, not even requiring a reboot at any point).

You can also create a new VDEV and add it to a zfs zpool, but the redundancy is in the VDEV. When you create a zpool of type RAIDZ2 (RAID-6), you are normally creating a single VDEV, and it is the VDEV that has the redundancy. You can add a new VDEV to a zpool, but the new VDEV must have its own redundancy. A clear article on this is at

https://louwrentius.com/the-hidden-cost-of-using-zfs-for-your-home-nas.html.

Controversy - File System

This became a major sticking point. Since OpenZFS is so tightly aligned with BSD (the license is incompatible with Linux), if we chose ZFS, it was logical to go with FreeBSD as our base. However, ZFS has pretty crummy standard software RAID (from what I can tell), so if we wanted to stay with mdadm and gain the flexibility if our requirements were off, we would need to run Devuan Linux.

Possible solution

We have used Xen virtualization for many years with good success. It is very reliable and appears to be efficient about its resource uses. So, a possibility was brought up. What if we build a Linux server, using mdadm and Xen virtualization, then pass the raid set *as a single device* directly to a FreeBSD virtual. The FreeBSD virtual could then create a ZFS file system on this. The added advantage would be that hast would only have to synchronize one device (the RAID device passed into the virtual).

Now, if we need to add disk space, we simply use mdadm to do it real time. Once the build was complete, zfs has the ability to notice that it has more space on a device than it is using and, in real time, extend itself to use the additional space (this was never tested, but is implied several places we looked during our research).

The big question was, what do we lose by placing our file system on top of a virtualization platform, on top of a software RAID set.

Testing the Solution

We ran some very simple tests. These were not exhaustive by any stretch of the imagination. We decided to build the system out as a FreeBSD/OpenZFS system, run some tests, then rebuild using Devuan ASCII/mdadm/Xen running a FreeBSD virtual. For the latter test, we purchased two low end SSD's and configured them as a RAID-1 to be used by the virtual for the boot volume.

Our tests were:

- bonnie++ with 5 iterations
- create a 100G file with data from /dev/rand
- Take the file created above and compress it using xz, using all processors.

Note: that we did no tests on the network speed, mainly because iSCSI does not use a lot of bandwidth from our original servers. Note: We decreased the amount of memory in the virtual to 28G, allocation the other 4G to the DOM0 itself

The assumption was that, if we only found a 10% efficiency loss, this was a viable idea. If we found it took twice as long to perform operations, it was not viable. Anything in between required a decision.

The results were not clear cut. While the reads were all pretty good, between 10 and 15% overhead, the writing from the virtual took 50% longer in some cases (the put_block median was 343719 K/sec, dropping to 217553 K/sec on the virtual).

This is acceptable in some cases. Actually, the ssd's are rated as 3.5 times as fast as the hard drives used in the original machine, so this particular machine would still, theoretically, be much faster. But, it was a downcheck.

Controversy - SSD's and TRIM

The other issue with using the hybrid Linux/FreeBSD setup was maintaining the SSD's in good working order. SSD's require some extra care when you are deleting files, and to maintain optimum performance, periodic TRIM's must be performed. These drives (the Samsungs) have garbage collection built into them, but the effectiveness is greatly enhanced by doing periodic TRIM's.

In this case, we have mdadm as the base, which abstracts the underlying structure to the ZFS file system. ZFS can not perform TRIM on the drives since the software raid hides the actual disks. The only solution is to underprovision the drives themselves.

By creating a partition that is 80% the size of the device and using that for your raid device, you leave 20% that is able to be handled quite efficiently by the built in garbage collection. However, you have then decreased your available capacity by 20%, meaning you need more drives to deliver the same functionality.

TRIM and Garbage Collection are described quite well in the article at https://arstechnica.com/gadgets/2015/04/ask-ars-my-ssd-does-garbage-collection-so-i-dont-need-trim-right/ if you want to know more. The article was written in 2015, but is still applicable today (2019)

Solution

There is no good solution in this case. For some services, it may be better to go with the hybrid Linux/mdadm/virtual, giving the flexibility to expand the system with no down time. In other cases, you may decide to use a bare metal setup, where your operating system is the only one on the machine.

In our case, we decided zfs needed to know the drives so it could keep them clean, and the overhead of virtualization was not worth the effort. That determined the operating system (FreeBSD).

We did make two adjustments based on these tests.

Memory

I have read many times people saying zfs is a memory intensive file system. We found proof of this during the tests. When the virtual was used, we decreased available memory from 32G to 28G, and bonnie++ aborted during the fourth test. Review of the logs showed swap space had been exhausted, so we added a 4G swap file and bonnie++ was able to complete the run. However, this means the zfs system was using swap in order to complete the process. It is possible, by adding additional memory, the results would have been much different. In our case, we have a schedule that must be met, so we were not able to run additional tests.

SSD's

By choosing to use SSD's instead of fast hard drives, we obligated ourselves to the added complexity necessary to maintain the efficiency of the SSD's. If we had chosen hard disks, we could have used the full disk space available and not had to concern ourselves with the abstraction of using mdadm. We got around this by changing our specifications of the project and purchasing additional drives to (hopefully) ensure we would not run out of disk space over the expected 5 year life of the system.

From:
https://wiki.linuxservertech.com/ - **Unix Server Tech Knowledge Base**

Permanent link:
https://wiki.linuxservertech.com/research/storage_server

Last update: **2019/07/02 08:08**