Project Report (May, 2017)

-- George (Zhi Qiao)

In May, our research emphasis on the performance analysis of ZFS recovery mechanism. We have installed 4 Seagate Barracuda 2TB hard drives with SATA interface, a 240 GB Intel data center version SSD (Intel DC3520) on our server. We also ordered 3 more Intel SSD, so we can setup two testbed, one with full HDD, and another with full SSD. The Intel DC 3520 SSD have 26 SMART attributes. While we use SSD for ZFS performance test, we also record its SMART status daily for our SSD study.

I/O Performance Benchmark

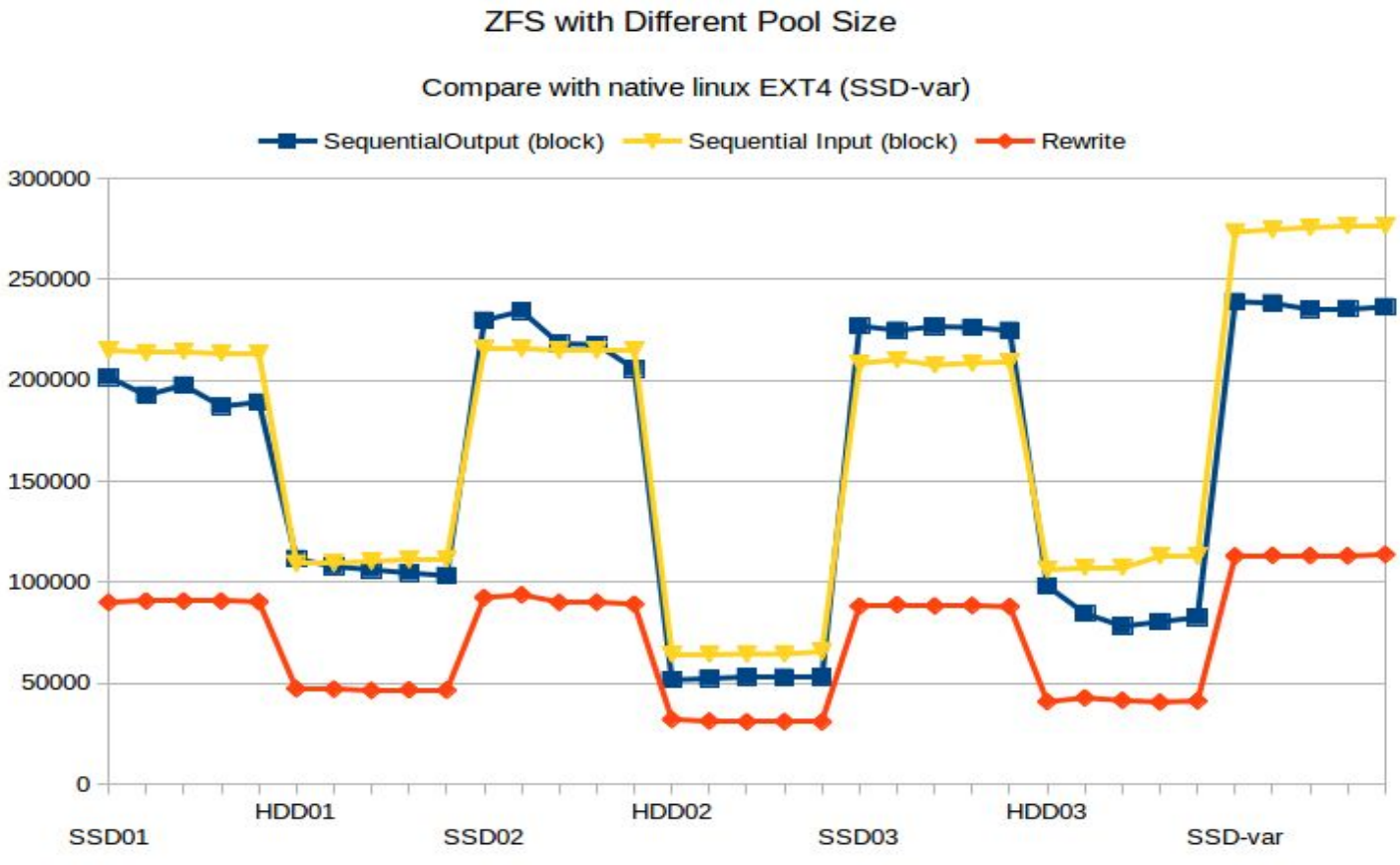
In order to create a baseline for our ZFS analysis model, our preliminary study towards the ZFS recovery performance starts with the I/O benchmark of ZFS on HDD and SSD. We use Bonnie++ in our test to benchmark different storage device with either ZFS or EXT4 as filesystem. We benchmark each disk one by one, then repeat the experiment 5 times to see if results are stable. Following is the chart showing the I/O performance difference between the ZFS and the EXT4 on different storage media: HDD and SSD.

Bonnie++ use *write()* to perform sequential output by block, and use *read()* to perform

sequential input. From the result, we can tell that 1) ZFS is slightly slower than EXT4 in the sequential write and read test, maybe this is caused by the overhead of ZFS itself; 2) SSD performs significantly better than HDD, as we expected 3) benchmark result stays very stable, so we can have more confidence in our performance modeling accuracy in future.

As we mention in last month's report, we try to use QEMU emulator to create several virtual disk with ZFS as filesystem. However, compared with physical disk, the benchmark results for virtual disk varies significantly and much slower. Hence indicate the overhead from QEMU is not neglectable, and the result cannot helps us build a reliable model. Therefore, we exclude QEMU emulator from our benchmark. It is still a nice tool to proof certain concept, for example compare different RAID level performance.

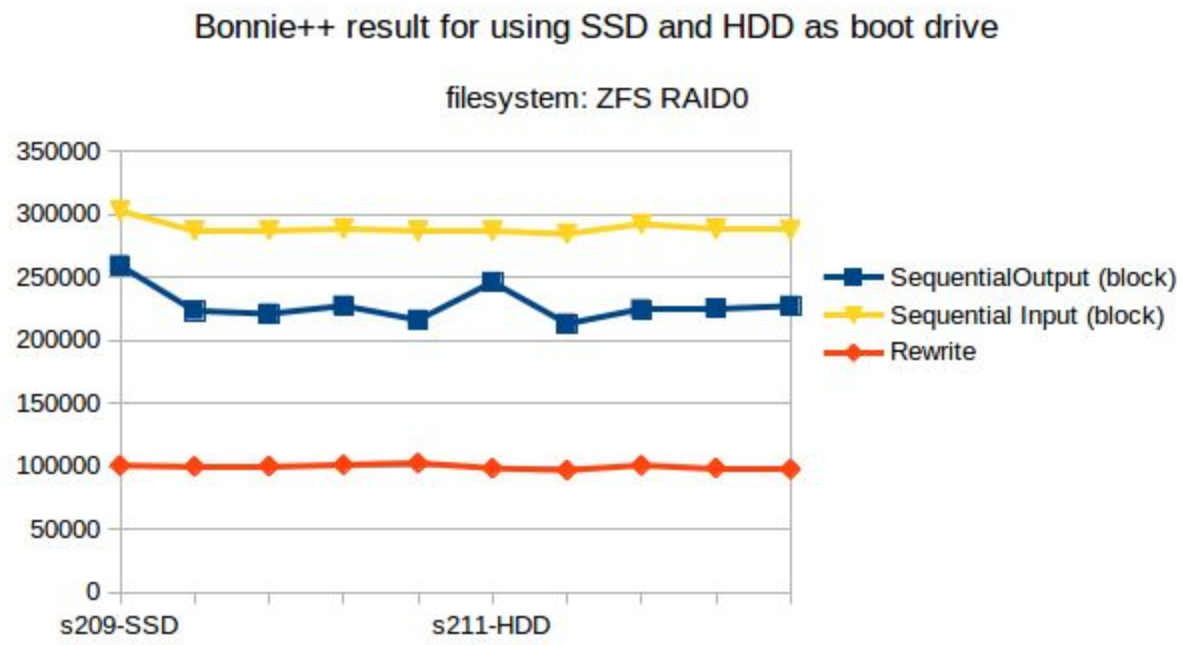
We also want to know if the different zpool size can have any impact on the performance of ZFS on HDD and SSD. We partition the disk into different size and use each single partition to create zpool. So pool SSD01 have 20GB, SSD02 have 40GB, and SSD03 have 80GB, similarly, HHD01 have 20GB, HHD02 have 40GB, HHD03 have 80GB. SSD-var indicate a EXT4 format partition hosting linux /var directory. We include it here just as a compare of ZFS and EXT4.



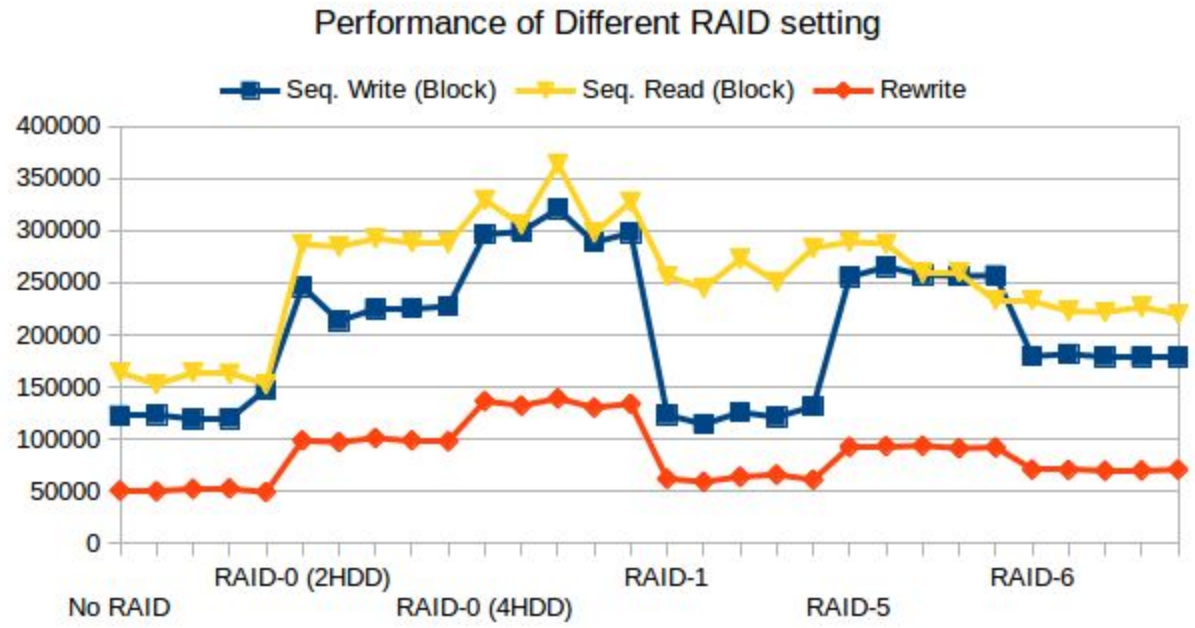
From the result, we can conclude that 1) for SSD, as pool size increase, write speed, or sequential output performance gets better, 2) but read speed, or sequential input performance stay the same. One reason for better write speed when pool becomes larger is due to the mechanism of modern SSD design, where larger pool size leads to more sparse distribution of the active data. So SSD data placement engine spend less time looking for empty blocks to allocate data. This feature explains the drop of SSD performance after its usage close to full. Meanwhile, the input performance stays the same while the pool size increase. This indicates another advantage of SSD, of which have no mechanical moving part involved in storage media, like HDD. Without physical spindle or magnetic disc, only SSD's address translation layers can determine the data lookup time, which is often a constant number. Therefore, SSD performance can be very stable for different data access pattern.

Our experiment discussed here still have some limitations. Since we use partitions on a single disk, the conclusion we got from this test cannot extended to ZFS on actual HDD or SSD with different capacities.

In order to see the performance impact of ZFS on different boot media, we setup two server with same spec, both have same type 2x HDD running ZFS. Only difference is the boot media: one server using Samsung EVO 850 SSD, the other one use Toshiba HDD with 5400 RPM as boot device. Two HDD are setup as RAID-0 or ZFS mirror. The benchmark result shows the boot drive have no significant impact on the file system performance.



To confirm the bonnie++ benchmark result are consistent in different RAID setting, we perform the next test: setting up server with 4 HDD and using SSD as boot drive; benchmark ZFS with different RAIDZ setting and using different bonnie++ parameters. By default, bonnie++ use testing set that double the system RAM size, to overcome the "caching effect" that may affect the accuracy of benchmark. Our testbed have 8GB RAM, so default testing set used by bonnie++ is 16GB. Following figure shows the result.

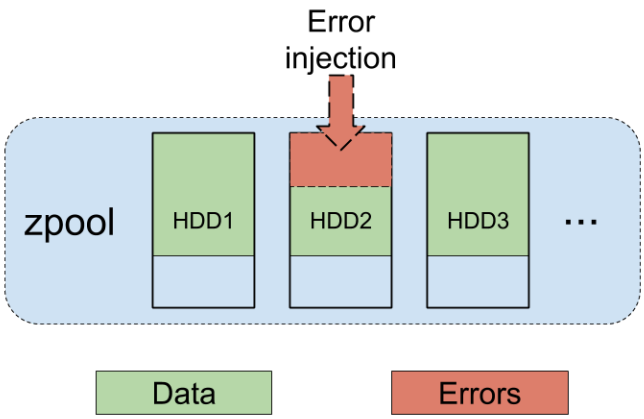


We have following observations: 1) RAID 1 have no write performance benefit than no RAID. Since each write needs to perform parallel on each disk in the mirror group. But read performance is much better in RAID-1 than in no RAID setting, as it share the aggregated read bandwidth. 2) more disk in RAID-0 strip, better performance. However, no replication is still the issue. 3) most of the time, bonnie++ can produce the stable result, however, in RAID-0 4 HDD and RAID-1 test, output have some fluctuations. This reminds us: we still have to run our test multiple times in order to get a convincing result.

Once we have the I/O performance as the baseline. We can move on to study the ZFS recovery performance.

ZFS Recovery/Resilver

Currently, we fill a zpool with random data, then inject zeros into physical disk directly. We record the recovery time reported by "*zpool status"* command. We tried this approach using QEMU emulator last month. The rebuild time of a 1GB virtual disk took less than two minutes. However, ZFS is ideal to handle large disk rebuild as it reports rebuild time in a "Day/Hour/Minutes" format. This resolution is too low for a 1GB disk. Moreover, result from an emulator might not be very convincing. So, we emphasis our work on actual disks with at least 500GB partition size. Figure above illustrate our approach.



1. Fill the pool with random number

First, we fill the zpool with random values to certain percentage, such as 25%, 50%, and 75% of the usage. Since we need to fill up the pool with several TB of data, we use *openssl PRNG engine* to create random data. This result much better performance compared with data generation using "dd" command (from 16hr for 1TB data using "dd if=/dev/urandom ", to about 1hr using openssl PRNG "openssl rand $((1<<40))").

2. Inject errors to drive directly

We then inject zeros to one of the physical disk. E.g.,: "dd if=/dev/zero of=/dev/sdb1/tmp bs=1G count=100". At this point, data in zpool should already been polluted. However, we have no control of where ZFS distributes the actual data, so we cannot tell how many data got corrupted. Moreover, ZFS will not constantly checking the checksum, so without reading the data or scrub the zpool, it will not find out the data have already corrupted. Therefore, we call the "zpool scrub" to "notify" ZFS for data recovery.

3. Let ZFS recover/resilver

Depends on how much data got corrupted, ZFS will either recover the data, or, report disk as failure if there are too many errors. We now use “dd” command to inject fix amount of zeros to control the level of corruption, and use the pipe viewer utility “pv” to inject a stream of zeros to flush the disk. In the prior case, “zpool scrub” command will report the number of corrupted bytes as comparison result with checksum, and start the recovery process immediately. After it finishes, we use “zpool status” to find out how long it took to recover the data from corruption. In the latter case, “pv” utility will fill the drive with zeros, and even flush away all the metadata that ZFS used as checksum. So it will report as disk failure. Once we re-format the disk or swap with another disk, involk “zpool replace”, zpool resilver process starts immediately. We also use “zpool status” to find out how long it took to resilver.

Recovery Performance Analysis

So far, we still working on the testing phase of collecting ZFS recovery time for different environment setting. List below are the factors we studied and will working on in next week.

- RAIDz (RAID-1, RAID-5, and RAID-6, RAID-7) recovery time, with same stripe size (1, 2, 3 disk as strips)
- RAIDz with different zpool utilization (25%, 50%, 75%)
- RAIDz with different stripe size (different number of disk)
- RAIDz with SSD as cache (SSD as cache, HDD as storage)
- RAIDz with SSD as storage media

Towards “Always on” Storage

We will compare result between 1) ZFS resilver performance with 2) proactive disk copy. However, ZFS only resilver “active data”, that means if the disk only used a small portion, ZFS resilver will be much faster than full disk copy. So we also try to find a way to proactively copy data that ZFS labelled as “active data”, so theoretically, could be faster than ZFS passively resilver performance.