

APPENDIX: ARTIFACT DESCRIPTION

A. Description

ZFS is a widely used file system, for example as the storage backend of Lustre. Speeding up the recovery and improving the reliability of ZFS is crucial for HPC storage systems. This artifact presents the test platform configuration, code, execution instructions, and experiment workflow that can be used to reproduce the experimental results.

B. Test Platform

- **Hardware:** Intel Xeon E5620, 120GB DDR3 DRAM (set to 70GB during I/O benchmarking), each JBOD server have 45 x 4TB Hitachi 3.5" SATA drives in Supermicro SC847 JBOD enclosure connected to storage server by SFF-8088 cable to LSI MegaRAID SAS controller. Each drive is configured to RAID 0 Write Through mode to minimize interference from the RAID card.
- **Runtime:** CentOS 7.5 with Linux kernel v3.10.0-862.
- **dRAID:** Source code is available at <https://github.com/thegreatgazoo/zfs> (draid branch).
- **Software dependence:** dRAID uses ZFS 0.7 and has not been merged into the latest official release. Build dRAID requires *spl* which is available at ZFS' official Github, <https://github.com/zfsonlinux/spl> (spl-0.7-release).
- **Benchmark:** We use IOZone v3.482 to test dRAID and raidz' I/O performance.

C. Installation

Ensure that *spl* is installed properly before running dRAID. Clone the source code from ZFS' official GitHub and use the spl-0.7 release.

```
$ git clone https://github.com/zfsonlinux/spl
$ git checkout spl-0.7 -release
$ ./autogen.sh
$ ./configure --enable-debug
$ make & make install
```

To install dRAID, clone the source code and checkout draid branch. Follow dRAID's HOWTO at <https://github.com/zfsonlinux/zfs/wiki/dRAID-HOWTO> for detailed instructions.

```
$ git clone https://github.com/thegreatgazoo/zfs.git
$ git checkout draid
$ ./autogen.sh
$ ./configure --enable-debug
$ make & make install
```

To use dRAID, load ZFS module using the following options.

```
$ modprobe zfs \
zfs_vdev_scrub_max_active=10 \
zfs_vdev_async_write_min_active=4 \
draid_debug_lvl=5
```

D. Configure and Create dRAID

A configuration file must be specified before creating dRAID. Use *draidcfg* command to specify the number of

data blocks *d*, parity blocks *p*, and spare drives *s*. They need to match with the total number of drives *n*. Then dRAID storage pool *tank* can be created by using the configuration file. Similar to the creation of RAIDZ, we need to specify the redundancy level of dRAID between draid1 and draid3.

```
$ draidcfg -p 3 -d 8 -s 3 -n 36 draid3_36.nvl
$ zpool create -f tank draid3 cfg=draid3_36.nvl
```

E. Performance Benchmarking

To compare with ZFS *raidz* and *mirroring*, we create raidz and dRAID separately using the same redundancy and equivalent configurations.

1) *I/O benchmarking:* Our scripts use *dd* command to gradually write random data to the zpool, and then save the measured performance metrics.

```
for i in {1..100}
do
dd if=/dev/urandom of=/tank/'date +%m%d%H%M%S'.dat bs=10M count=27000
done
```

IOZone generates and measures a variety of file operations. Place the *IOZone* executable in the *zpool* directory, and then run the benchmark. The results are saved in an Excel file. Note that in CentOS 7.5 (Linux kernel v3.10.0-862), ZFS' directory does not allow to run *ls* and throws error: *ls: reading directory .: Not a directory*. Changing to kernel v3.10.0-693 can fix the problem.

```
$ ./iozone -Rab iozone_draid3_36hdd.xls
-y 16384 -n 16384 -g 140G
```

2) *Error injection:* The partition table and RAID configuration files are usually located in the first and last sections of a disk drive. We flush the area on the target drive by *zeros* so that ZFS detects the target drive as failed. This approach is hardware dependent. A more general approach is to use a virtual machine to make the drive inaccessible.

```
$ dd bs=512 if=/dev/zero of=/dev/sdx \
count=$((2048*500)) seek=$((`blockdev
--getsz /dev/sdx` - 2048*500 ))
$ dd bs=512 if=/dev/zero of=/dev/sdx \
count=$((2048*500))
```

3) *dRAID rebuild vs. RAIDZ resilvering:* Write random data to the storage pool so that dRAID and RAIDZ can recover the pool later. After injecting errors to the target drive, we call *zpool scrub tank* to trigger ZFS to detect the corruption in the "failed" drive. dRAID will rebuild data on spare drives.

```
$ zpool replace tank sdx '%draid3-0-s0'
```

RAIDZ resilvers data to spare drives.

```
$ zpool replace -f tank sdx sdo
```

Use *zpool status tank* to get more information of the data recovery process. ZFS reports how long it takes to recover data and the amount of data recovered.