dRAID: Declustered RAID for ZFS

Installation and Configuration Guide
High Performance Data Division

INTEL FEDERAL, LLC PROPRIETARY
September 2017

Generated under Argonne Contract number: B609815

DISTRIBUTION STATEMENT: None Required

Company Name: Intel Federal, LLC

Company Address: 4100 Monument Corner Drive, Suite 540 Fairfax, VA 22030

Technical Lead: (Name, E-mail, Phone) _Al Gara, alan.gara@intel.com, 408-765-0996

Contract Administrator: (Name, E-mail, Phone) Aaron Matzkin, aaron.matzkin@intel.com, 503-712-0833

Program Manager: (Name, E-mail, Phone) Jacob Wood, jacob.r.wood@intel.com, 503-264-2219

Document Revision History

Revision Number	Date	Comments
0.8	September 2017	Initial version.

Contents

Tables

Figures

1 Introduction

In large-scale storage configurations needed to meet the IO requirements of future HPC systems, disk failures are inevitable and viewed as a normal incident rather than an exceptional event.

When disk failures happen, it is important that RAID parity reconstruction complete as quickly as possible. Shorter rebuild times significantly reduce exposure to multiple concurrent disk failures, which could lead to data loss. It is also important that the RAID rebuilding process minimally affect the application IO. A drop in IO performance would cause applications to run longer or even fail to complete in their allotted time window.

The process of rebuilding a “traditional” RAID array, when replacing a failing/failed drive by a new one, consists of reading all the data, block-by-block, on all the surviving disks in the array, reconstructing the original blocks of the failed drive, and then writing the reconstructed data to the replacement drive. After this process is complete, the array is restored to its original full redundancy.

In ZFS, there is a similar and equivalent process, called resilvering, which is implemented differently from traditional RAID reconstruction, as volume management is a built-in part of ZFS. This process starts by traversing the ZFS block pointer tree to discover all the blocks of the ZFS pool that were affected by the failed drive. Upon reaching one of these blocks, the block is read, or reconstructed if necessary from the redundant/parity information, its checksum is verified, and the missing data or parity from the failed drive is written to free blocks on a new drive.

In both cases, the speed of rebuilding or resilvering is bounded by the write throughput of a single replacement drive. As a result, the total resilver time will grow at least linearly (often much worse) with drive capacity. As drive capacity continues to grow with little increase in drive throughput, rebuild time can increase significantly. For example, it would take about 27 hours to rebuild a 10TB drive at 100MB/s. Since idle time is rare, a drive failure and subsequent rebuild process can significantly affect system performance.

Parity declustered RAID (dRAID) for ZFS distributes data, parity, and spare capacity across all drives in a pool so that they all participate in the rebuild process equally. Since the pool is many times larger than the redundancy group size, aggregate read performance during reconstruction is correspondingly increased. In addition, since reconstructed data is written to spare space distributed across all drives, the bottleneck of having to write a single replacement drive to restore redundancy is eliminated.

Though declustered RAID for ZFS uses the existing RAIDZ code to calculate parity and reconstruct lost blocks, we call the solution dRAID to make clear the distinction between the layout of data, parity, and spare space in this design and the layout of data and parity with existing RAIDZ. The dRAID solution makes possible a mechanism to further speed up recovery after a drive failure by decoupling the recreation of redundancy from the verification of the recreated blocks to ensure that no drive in the storage array is idle during the rebuild.

1.1 Terms used in this Document

The following terms and abbreviations are used in this document.

Term	Definition
ZFS	A combined file system and logical volume manager.
RAIDZ	A generic term to refer to ZFS RAIDZ1, RAIDZ2, RAIDZ3, and mirror, when there is no need to distinguish between them. Otherwise, the more specific terms are used. The generic term RAID is also used when there is no need to distinguish between traditional RAID and RAIDZ.
VDEV	A "virtual device" describes a single device or a collection of devices organized according to certain performance and fault characteristics. ZFS currently supports the following VDEV types: disk, file, mirror, RAIDZ, spare, log, and cache.
ZFS Pool	Unlike traditional file systems which reside on single devices and thus require a volume manager to use more than one device, ZFS file systems are built on top of virtual storage pools called ZFS pools. A ZFS Pool is a constructed of a set of VDEVs.
dRAID	A modification of RAIDZ, as defined in this document, to implement declustered RAID for ZFS using fixed stripe-width redundancy groups to improve RAIDZ resilver speed. This is a generic term that can refer to ZFS dRAID1 (single parity), ZFS dRAID2 (double parity), ZFS dRAID3 (triple parity), and ZFS dRAIDM (mirror). Though dRAID will use the existing RAIDZ code to calculate parity and reconstruct lost blocks, we call the solution dRAID to make clear the distinction between the layout of data, parity, and spare space in this design and the layout of data and parity with existing RAIDZ.
Drive Slice	All drives in a dRAID VDEV are divided into equal sized units called slices. A slice is the basic unit of parity declustering. Slice size must be a multiple of hardware sector size of the drive.
Metaslab	An allocation region in a VDEV. ZFS divides a top-level VDEV into equal-sized regions called metaslabs. A ZFS block cannot cross metaslab boundary.
Permutation	Permutation and developed permutation is derived from the base permutation.
Redundancy Group	The redundancy group is composed of data and parity units that RAIDZ generates from the file block it receives from ZFS. Reconstruction of the group is possible if one or more (depending on the RAIDZ type) of its units are unreadable.
Resilver	The process of reconstructing data/parity on a failed drive in a RAIDZ group to a replacement drive, or failed drive in a dRAID group to spare space.
Scrub	The process of examining all ZFS blocks in a pool to verify block checksums. For replicated VDEVs (mirror, RAIDZ, or dRAID), ZFS automatically repairs any damage discovered.
Spacemap	Persistent on-disk data structure that keeps track of allocated space in a metaslab. There is one spacemap for each metaslab.

Term	Definition
Spare Rebalance	The process of copying reconstructed data/parity from previous spare blocks to a replacement drive so that distributed spare blocks become available again.
Spare Space	This is spare capacity distributed over all drives in a dRAID VDEV, reserved for recovery. For the sake of simplicity hereafter in this document, N spare drives is used as a shorthand for distributed spare space with sufficient capacity to rebuild data on N failed drives.
Uberblock	A VDEV label contains an array of uberblocks. The uberblock is the portion of the label containing information necessary to access the contents of the pool. Only one uberblock in the pool is active at any point in time. The uberblock with the highest transaction group number and valid SHA-256 checksum is the active uberblock.
Unit	A unit is a portion of a redundancy group written to a drive slice. A redundancy group is composed of data and parity units.
VDEV Label	Each physical VDEV within a ZFS pool contains four copies of a 256KB structure called a VDEV label, two at the beginning of the VDEV and two at the end. The VDEV label contains information describing this particular physical VDEV and all other VDEVs which share a common top-level VDEV as an ancestor.
DVA	The Data Virtual Address is the ZFS notion of block address. It consists of two parts: VDEV, and offset. It determines the physical location of a ZFS block on a top-level VDEV.
ZED	ZFS Event Daemon monitors events generated by the ZFS kernel module. When a zevent (ZFS Event) is posted, ZED will run any ZEDLETs (ZFS Event Daemon Linkage for Executable Tasks) that have been enabled for the corresponding zevent class.

1.2 Additional Documentation

Refer to the following documentation for architecture and description:

Document	Location
Scope Statement	
Solution Architecture	

1.3 Software Requirements

- ZFS on Linux version 0.8.0.
- Lustre* version 2.10
  NOTE: While Lustre is not required for dRAID to be used in a ZFS environment, it is required for some of the features described in this document.

1.4 Hardware Requirements

The hardware used must be compliant with the minimum RAIDZ requirements (Minimum Drives= (5+1) single parity).

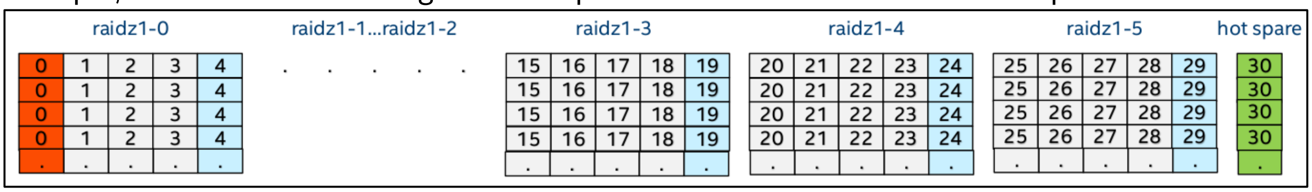
2 Configuring dRAID for ZFS

2.1 Introduction

This chapter describes the setup and configuration of dRAID for ZFS.

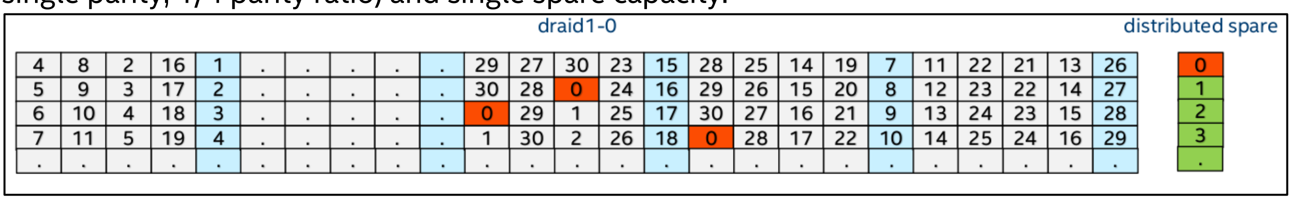
2.1.1 raidz vs dRAID

ZFS users are most likely very familiar with raidz already, so a comparison with dRAID may help. The illustrations below are simplified, but sufficient for the purpose of a comparison. For example, 31 drives can be configured as a zpool of six raidz1 VDEVs and a hot spare:



As shown above, if drive 0 fails and is replaced by the hot spare, only five out of the 30 surviving drives will work to resilver: drives 1-4 read, and drive 30 writes.

The same 30 drives can be configured as 1 dRAID1 VDEV of the same level of redundancy (i.e. single parity, 1/4 parity ratio) and single spare capacity:



The drives are shuffled in a way that, after drive 0 fails, all 30 surviving drives will work together to restore the lost data/parity:

- All 30 drives read, because unlike the raidz1 configuration shown above, in the dRAID1 configuration the neighbor drives of the failed drive 0 (i.e. drives in a same data+parity group) are not fixed.
- All 30 drives write, because now there is no dedicated spare drive. Instead, spare blocks come from all drives.

To summarize:

- Normal application IO: dRAID and raidz are very similar. There is a slight advantage in dRAID, since there is no dedicated spare drive that is idle when not in use.
- Restore lost data/parity: for raidz, not all surviving drives will work to rebuild, and in addition, it is bounded by the write throughput of a single replacement drive. For dRAID, the rebuild speed will scale with the total number of drives because all surviving drives will work to rebuild.

The dRAID VDEV must shuffle its child drives in a way that regardless of which drive has failed, the rebuild IO (both read and write) will distribute evenly among all surviving drives, so the rebuild speed will scale. The exact mechanism used by the dRAID VDEV driver is beyond the scope of this simple introduction here. If interested, please refer to the recommended readings in the next section.

2.1.2 Recommended Reading

Parity declustering (the term used for shuffling drives) has been an active research topic, and many papers have been published in this area. The Permutation Development Data Layout is a recommended paper to begin. The dRAID VDEV driver uses a shuffling algorithm loosely based on the mechanism described in this paper.

2.2 Using dRAID

The dRAID code will be included in the ZFS on Linux distribution. Build spl and zfs with `configure`, and install. Then load the zfs kernel module with the following options:

- zfs_vdev_scrub_min_active=2
  zfs_vdev_scrub_max_active=10
  zfs_vdev_async_write_min_active=8:
  These options help dRAID rebuild performance.
- draid_debug_lvl=5:
  This option controls the verbosity level of dRAID debug traces, which is very useful for troubleshooting.

2.2.1 Create a dRAID VDEV

Unlike a raidz VDEV, before a dRAID VDEV can be created, a configuration file must be created with the `draidcfg` command:

```
# draidcfg -p 1 -d 4 -s 2 -n 17 17.nvl
Not enough entropy at /dev/random: read -1, wanted 8.
Using /dev/urandom instead.
 Worst ( 3 x  5 +  2) x   544: 0.882
Seed chosen: f0cbfeccac3071b0
```

The command in the example above creates a configuration for a 17-drive dRAID1 VDEV with four data blocks per strip and two distributed spares, and saves it to file *17.nvl*.

Options:

- p: parity level, can be 1, 2, or 3.
- d: # data blocks per stripe.
- s: # distributed spare
- n: total # of drives
- It's required that: (n - s) % (p + d) == 0

Note:

- Errors like "Not enough entropy at /dev/random" are harmless

- In the future, the *draidcfg* may get integrated into *zpool create* so there would be no separate step for configuration generation.

The configuration file is binary, to examine the contents:

```
# draidcfg -r 17.nvl
dRAID1 vdev of 17 child drives: 3 x (4 data + 1 parity) and 2 distributed
spare
Using 32 base permutations
   1,12,13, 5,15,11, 2, 6, 4,16, 9, 7,14,10, 3, 0, 8,
   0, 1, 5,10, 8, 6,15, 4, 7,14, 2,13,12, 3,11,16, 9,
   1, 7,11,13,14,16, 4,12, 0,15, 9, 2,10, 3, 6, 5, 8,
   5,16, 3,15,10, 0,13,11,12, 8, 2, 9, 6, 4, 7, 1,14,
   9,15, 6, 8,12,11, 7, 1, 3, 0,13, 5,16,14, 4,10, 2,
  10, 1, 5,11, 3, 6,15, 2,12,13, 9, 4,16,14, 0, 7, 8,
  10,16,12, 7, 1, 3, 9,14, 5,15, 4,11, 2, 0,13, 8, 6,
   7,12, 4,13, 6,11, 9,15,14, 2,16, 3, 0, 1,10, 5, 8,
  10, 5, 8, 2, 1,11,16,15,12, 3,13, 4, 0, 7, 9, 6,14,
   1, 6,15, 0,14, 5, 9,11, 8,16,10, 2,13,12, 3, 4, 7,
  14, 4, 2, 0,12, 7, 3, 6, 8,13,10, 1,11,16,15, 9, 5,
   6,14, 8,10, 1, 0,15, 4, 5, 3,16,13, 9,12, 2, 7,11,
  13, 5, 8,14, 1,10,16,11,15, 7, 0,12, 2, 9, 4, 6, 3,
   9, 6, 3, 7,15, 1, 4, 8,14, 5, 0, 2,16,10,12,11,13,
  12, 0, 6, 7, 1, 9,14, 8,11,16, 4, 2,13,15, 3, 5,10,
  14, 6,12,10,15,13, 7, 0, 3,16, 5, 9, 2, 8, 4,11, 1,
  15,16, 8,13, 6, 4, 7,11, 1, 2,14,12,10, 5, 9, 3, 0,
   0,11,10,14,12, 1,16, 3,13, 9, 5, 7, 2, 4, 6,15, 8,
   2,10,12, 4, 3, 5,15, 1,11, 0, 7,13, 6, 9,14, 8,16,
  11, 8,16,12, 6,13,10, 9, 2, 7, 3, 4, 5, 0,14,15, 1,
   4,16,12,15,14, 3, 7, 1, 9,10, 6, 8,11, 0,13, 2, 5,
   5,16,13,11, 4, 6, 7,12, 0, 9,15, 1,14, 3, 8,10, 2,
  12, 6, 7, 0,10,15, 8, 2,16,14,11, 1, 4, 5, 9,13, 3,
   8, 4, 1,13, 6, 5, 0,15, 7, 3,11,14,16, 9,10,12, 2,
  16,14,15, 2,10,11, 6,13, 4, 9, 8, 0, 5,12, 3, 1, 7,
   9, 6, 8, 3,12,14,16,13,11,10, 4, 5, 7,15, 2, 0, 1,
   3, 9,15, 0, 7, 1, 8,11,12, 2,10, 6,13,16, 5,14, 4,
   0,14, 6,16, 1,10, 9,15,12, 8,11, 3, 2, 7,13, 5, 4,
  12,13, 9, 5,11, 6, 3, 4,14,10, 1, 7, 8, 2, 0,16,15,
  16, 9, 0, 2, 3,10, 1,11, 6, 4,13,12,14, 7, 5,15, 8,
  16, 9, 6, 0, 1, 4,11,14,12, 3, 2,15,13,10, 5, 8, 7,
   7, 8,11,14,10, 6,15,13, 1, 4,16, 9, 2, 3, 0,12, 5,
```

Now a dRAID VDEV can be created using the configuration. The only difference from a normal *zpool create* is the addition of a configuration file in the VDEV specification:

```
# zpool create -f tank draid1 cfg=17.nvl sdd sde sdf sdg sdh sdi sdj sdk sdl
sdm sdn sdo sdp sdq sdr sds sdt
```

Note:

- The total number of drives must equal the *-n* option of *draidcfg*.

- The parity level must match the *-p* option (for example, use draid3 for *draidcfg -p 3*).

When the numbers do not match, *zpool create* will fail but with a generic error message, which can be confusing.

Now the dRAID VDEV is online and ready for IO:

```
# zpool status
  pool: tank
 state: ONLINE
config:

        NAME            STATE     READ WRITE CKSUM
        tank            ONLINE       0     0     0
          draid1-0      ONLINE       0     0     0
            sdd         ONLINE       0     0     0
            sde         ONLINE       0     0     0
            sdf         ONLINE       0     0     0
            sdg         ONLINE       0     0     0
            sdh         ONLINE       0     0     0
            sdu         ONLINE       0     0     0
            sdj         ONLINE       0     0     0
            sdv         ONLINE       0     0     0
            sdl         ONLINE       0     0     0
            sdm         ONLINE       0     0     0
            sdn         ONLINE       0     0     0
            sdo         ONLINE       0     0     0
            sdp         ONLINE       0     0     0
            sdq         ONLINE       0     0     0
            sdr         ONLINE       0     0     0
            sds         ONLINE       0     0     0
            sdt         ONLINE       0     0     0
        spares
          $draid1-0-s0  AVAIL
          $draid1-0-s1  AVAIL
```

There are two logical spare VDEVs shown above at the bottom:

- The names begin with a '$' followed by the name of the parent dRAID VDEV.
- These spare are logical, made from reserved blocks on all the 17 child drives of the dRAID VDEV.
- Unlike traditional hot spares, the distributed spare can only replace a drive in its parent dRAID VDEV.

The dRAID VDEV behaves just like a raidz VDEV of the same parity level (IO to/from it, scrub it, fail a child drive and it would operate in degraded mode).

2.2.2 Sequential Rebuild

When there is a bad/offlined/failed child drive, the dRAID VDEV supports a completely new mechanism to reconstruct lost data/parity, in addition to the resilver. First of all, resilver is still supported - if a failed drive is replaced by another physical drive, the resilver process is used to reconstruct lost data/parity to the new replacement drive, which is the same as a resilver in a raidz VDEV.

But if a child drive is replaced with a distributed spare, a new process called rebuild is used instead of resilver:

```
# zpool offline tank sdo
# zpool replace tank sdo '$draid1-0-s0'
# zpool status
  pool: tank
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
  scan: rebuilt 2.00G in 0h0m5s with 0 errors on Fri Feb 24 20:37:06 2017
config:

        NAME                STATE     READ WRITE CKSUM
        tank                DEGRADED     0     0     0
          draid1-0          DEGRADED     0     0     0
            sdd             ONLINE       0     0     0
            sde             ONLINE       0     0     0
            sdf             ONLINE       0     0     0
            sdg             ONLINE       0     0     0
            sdh             ONLINE       0     0     0
            sdu             ONLINE       0     0     0
            sdj             ONLINE       0     0     0
            sdv             ONLINE       0     0     0
            sdl             ONLINE       0     0     0
            sdm             ONLINE       0     0     0
            sdn             ONLINE       0     0     0
            spare-11        DEGRADED     0     0     0
              sdo           OFFLINE      0     0     0
              $draid1-0-s0  ONLINE       0     0     0
            sdp             ONLINE       0     0     0
            sdq             ONLINE       0     0     0
            sdr             ONLINE       0     0     0
            sds             ONLINE       0     0     0
            sdt             ONLINE       0     0     0
        spares
          $draid1-0-s0      INUSE     currently in use
```

```
          $draid1-0-s1      AVAIL
```

The scan status line of the *zpool status* output now says *"rebuilt"* instead of *"resilvered"*, because the lost data/parity was rebuilt to the distributed spare by a brand new process called *"rebuild"*. The main differences from *resilver* are:

- The rebuild process does not scan the whole block pointer tree. Instead, it only scans the spacemap objects.
- The IO from rebuild is sequential, because it rebuilds metaslabs one by one in sequential order.
- The rebuild process is not limited to block boundaries. For example, if 10 64K blocks are allocated contiguously, then rebuild will fix 640K at one time. So rebuild process will generate larger IOs than resilver.
- For all the benefits above, there is one price to pay. The rebuild process cannot verify block checksums, since it does not have block pointers.
- Moreover, the rebuild process requires support from on-disk format, and **only** works on dRAID and mirror VDEVs. Resilver, on the other hand, works with any VDEV (including dRAID).

Although the rebuild process creates larger IOs, the drives will not necessarily see large IO requests. The block device queue parameter */sys/block/*/queue/max_sectors_kb* must be tuned accordingly. However, since the rebuild IO is already sequential, the benefits of enabling larger IO requests might be marginal.

At this point, redundancy has been fully restored without adding any new drive to the pool. If another drive is offlined, the pool is still able to do IO:

```
# zpool offline tank sdj
# zpool status
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
  scan: rebuilt 2.00G in 0h0m5s with 0 errors on Fri Feb 24 20:37:06 2017
config:

        NAME                STATE     READ WRITE CKSUM
        tank                DEGRADED     0     0     0
          draid1-0          DEGRADED     0     0     0
            sdd             ONLINE       0     0     0
            sde             ONLINE       0     0     0
            sdf             ONLINE       0     0     0
            sdg             ONLINE       0     0     0
            sdh             ONLINE       0     0     0
            sdu             ONLINE       0     0     0
```

```
          sdj             OFFLINE      0     0     0
          sdv             ONLINE       0     0     0
          sdl             ONLINE       0     0     0
          sdm             ONLINE       0     0     0
          sdn             ONLINE       0     0     0
          spare-11        DEGRADED     0     0     0
            sdo           OFFLINE      0     0     0
            $draid1-0-s0  ONLINE       0     0     0
          sdp             ONLINE       0     0     0
          sdq             ONLINE       0     0     0
          sdr             ONLINE       0     0     0
          sds             ONLINE       0     0     0
          sdt             ONLINE       0     0     0
      spares
        $draid1-0-s0      INUSE     currently in use
        $draid1-0-s1      AVAIL
```

As shown above, the *draid1-0* VDEV is still in *DEGRADED* mode although two child drives have failed and it's only single-parity. Since the *$draid1-0-s1* is still *AVAIL*, full redundancy can be restored by replacing *sdj* with it, without adding new drive to the pool:

```
# zpool replace tank sdj '$draid1-0-s1'
# zpool status
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
  scan: rebuilt 2.13G in 0h0m5s with 0 errors on Fri Feb 24 23:20:59 2017
config:

        NAME                STATE     READ WRITE CKSUM
        tank                DEGRADED     0     0     0
          draid1-0          DEGRADED     0     0     0
            sdd             ONLINE       0     0     0
            sde             ONLINE       0     0     0
            sdf             ONLINE       0     0     0
            sdg             ONLINE       0     0     0
            sdh             ONLINE       0     0     0
            sdu             ONLINE       0     0     0
            spare-6         DEGRADED     0     0     0
              sdj           OFFLINE      0     0     0
              $draid1-0-s1  ONLINE       0     0     0
            sdv             ONLINE       0     0     0
            sdl             ONLINE       0     0     0
            sdm             ONLINE       0     0     0
```

```
            sdn             ONLINE       0     0     0
            spare-11        DEGRADED     0     0     0
              sdo           OFFLINE      0     0     0
              $draid1-0-s0  ONLINE       0     0     0
            sdp             ONLINE       0     0     0
            sdq             ONLINE       0     0     0
            sdr             ONLINE       0     0     0
            sds             ONLINE       0     0     0
            sdt             ONLINE       0     0     0
        spares
          $draid1-0-s0      INUSE     currently in use
          $draid1-0-s1      INUSE     currently in use
```

Again, full redundancy has been restored without adding any new drive. If another drive fails, the pool will still be able to handle IO, but there'd be no more distributed spare to rebuild (both are in *INUSE* state now). At this point, there's no urgency to add a new replacement drive because the pool can survive yet another drive failure.

2.2.2.1 Dynamic Rebuild Throttling

The rebuild process may delay zio according to the ZFS options `spa_vdev_scan_delay` and `spa_vdev_scan_idle`, which works in a similar way as options used by resilver `zfs_scan_idle` and `zfs_resilver_delay`. Moreover, when a dRAID VDEV has lost all redundancy, e.g. a draid2 with 2 faulted child drives, the rebuild process will go full speed by ignoring `spa_vdev_scan_delay` and `spa_vdev_scan_idle` altogether because the VDEV is now in critical state.

After delaying, the rebuild zio is issued using priority `ZIO_PRIORITY_SCRUB` for reads and `ZIO_PRIORITY_ASYNC_WRITE` for writes. Therefore the options that control the queuing of these two IO priorities will affect rebuild zio as well, for example `zfs_vdev_scrub_min_active`, `zfs_vdev_scrub_max_active`, `zfs_vdev_async_write_min_active`, and `zfs_vdev_async_write_max_active`.

2.2.3 dRAID-aware Spare Space Rebalancing

Distributed spare space can be made available again by simply replacing any failed drive with a new drive. This process is called *rebalance* which is essentially a *resilver*:

```
# zpool replace -f tank sdo sdw
# zpool status
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
  scan: resilvered 2.21G in 0h0m58s with 0 errors on Fri Feb 24 23:31:45 2017
config:
```

```
        NAME                STATE     READ WRITE CKSUM
        tank                DEGRADED     0     0     0
          draid1-0          DEGRADED     0     0     0
            sdd             ONLINE       0     0     0
            sde             ONLINE       0     0     0
            sdf             ONLINE       0     0     0
            sdg             ONLINE       0     0     0
            sdh             ONLINE       0     0     0
            sdu             ONLINE       0     0     0
            spare-6         DEGRADED     0     0     0
              sdj           OFFLINE      0     0     0
              $draid1-0-s1  ONLINE       0     0     0
            sdv             ONLINE       0     0     0
            sdl             ONLINE       0     0     0
            sdm             ONLINE       0     0     0
            sdn             ONLINE       0     0     0
            sdw             ONLINE       0     0     0
            sdp             ONLINE       0     0     0
            sdq             ONLINE       0     0     0
            sdr             ONLINE       0     0     0
            sds             ONLINE       0     0     0
            sdt             ONLINE       0     0     0
        spares
          $draid1-0-s0      AVAIL
          $draid1-0-s1      INUSE     currently in use
```

Note that the scan status now shows *"resilvered"*. In addition, the state of *$draid1-0-s0* has become *AVAIL* again. Since the resilver process checks block checksums, it makes up for the lack of checksum verification during previous rebuild.

The dRAID1 VDEV in this example shuffles three (4 data + 1 parity) redundancy groups to the 17 drives. For any single drive failure, only about 1/3 of the blocks are affected (and should be resilvered/rebuilt). The rebuild process is able to avoid unnecessary work, but the resilver process by default will not. The rebalance (which is essentially resilver) can speed up a lot by setting module option *zfs_no_resilver_skip* to 0. This feature is turned off by default because of issue https://github.com/zfsonlinux/zfs/issues/5806.

2.2.4 Troubleshooting

Please report bugs to the dRAID project, as long as the code is not merged upstream. The following information would be useful:

- dRAID configuration, i.e. the *.nvl file created by *draidcfg* command.
- Output of *zpool events -v*

- dRAID debug traces, which by default goes to *dmesg* via *printk()*. The dRAID debugging traces can also use *trace_printk()*, which is more preferable but unfortunately GPL only. It can be enabled by editing the META file to change the license (strictly for debugging only

2.3 Administration of dRAID for ZFS

2.3.1 Introduction

This chapter describes the administration of the dRAID for ZFS implementation

2.3.2 Command Line Interface

The ZFS block allocation code has been refactored to accommodate support for multiple metadata classes backed by one or more virtual devices. Fine grain accounting, by class distinction, was added to each runtime metaslab instance and is persistently stored in the on-disk space map object. The ZFS ztest tool was modified to exercise new metadata allocation code paths (section 5).

The CLI implementation for administering metadata classes is a set of extensions to the existing zpool(1) and zdb(1) commands. The augmented CLI allows metadata classes to be specified on pool create and later when adding additional VDEVs to a pool. In the CLI commands that display VDEV configurations, we added a class info summary to differentiate a VDEV's classes.

It is worth noting that in the ZFS CLI there are several methods of exposing the pool configuration, and metadata isolation had to be adapted for each method (Figure 2-1). Testing uncovered that some of the testing tools assumed a predetermined format for list log specific devices and we had to revert our generalizations for the specific case of logs (class = 'log').

zpool status <pool>
zpool list -v <pool>
zpool iostat -v <pool>
zpool import
zpool create -n <pool> <vdev>...
zpool add -n <pool> <vdev>
zdb -C <pool>
zdb -s <pool>

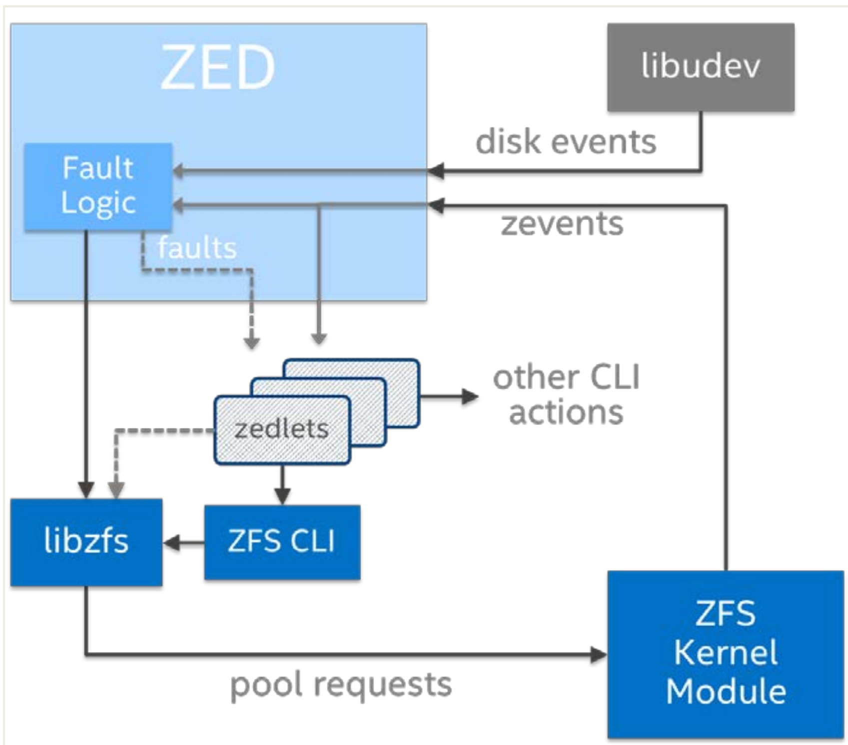
Figure 2-1. ZFS Commands Modified for Metadata Isolation

2.4 Tuning dRAID for ZFS

3 ZED Fault Handling

The Fault Management Architecture (FMA) has been migrated from OpenZFS to the Linux ZFS Event Daemon (ZED). Before this integration, ZED received events from the ZFS kernel module and called scripts, called zedlets, to respond to specific events. The addition of FMA allowed ZED to refine event processing so zedlets would only be called for specific faults (Figure 3-1). FMA provides critical fault logic to ZED and enables automatic rebuild and rebalance for dRAID and RAIDZ VDEVs.

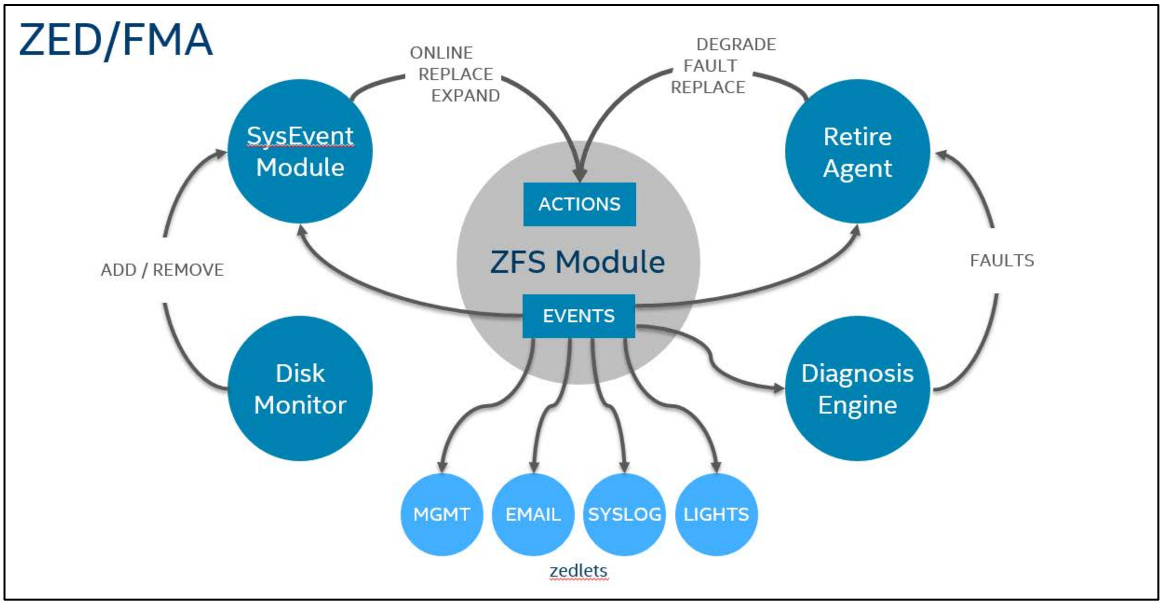
Figure 3-1. ZED Architecture



3.1 Introduction

The Fault Management Architecture (Figure 3-2) consists of four components-- the Diagnosis Engine, the Retire Agent, the SLM (syseventd loadable module) and the Disk Event Monitor -- that evaluate and act upon storage events and faults. The Diagnosis Engine receives events from ZFS and evaluates faults for the VDEVs in the system. The Retire Agent responds to diagnosed faults and, if necessary, initiates automatic rebuilds. The Agent will notify the ZFS kernel of the change in the VDEV state (degraded or faulted). When the Disk Monitor encounters a drive replacement, the event will be received by the Retire Agent, which will initiate rebalancing data from the surviving drives to replace the new drive for the failed one.

Figure 3-2. ZED FMA Components



3.2 Spare Device Matching

With the addition of Allocation Classes and dRAID, the nature of spare devices has changed. Before these two features were added, all spares were essentially equal and any spare could be used to effectively replace any drive in the pool. With the introduction of dRAID, however, spare drives are no longer physical devices. With the introduction of the special allocation classes, additional characteristics, such as size and type of drive, are important in selecting a spare.

A spare drive in a dRAID is a virtual spare composed of blocks randomly scattered across all of the physical devices in the pool. The nature of a dRAID spare means that a zpool with multiple dRAID VDEVs can only replace a failed drive with a spare that shares the same dRAID parent VDEV. A normal physical drive can also be used as a spare for a dRAID VDEV, but doing so will trigger a resilver of the pool. Since resilvering is a significantly slower operation than a dRAID sequential rebuild, using a normal drive defeats the purpose of using dRAID and should only be done if a dRAID distributed spare is unavailable. To address these concerns the Retire Agent in ZED will only attempt to spare-in a drive to a dRAID VDEV if the spare VDEV is a distributed spare ($draid) that has the same parent identifier as the dRAID VDEV it is being spared into.

Metadata Isolation (Section 4) uses a special allocation class to save ZFS metadata and small block data to metaslabs segregated from the RAID VDEV or to physical disks in a dedicated VDEV. When a dedicated tier is used, a different type of physical disk may be used to back this tier. For example, for a metadata heavy workload, a dedicated pair of SSDs may be used and a spare for this tier also be an SSD of similar size. The not just match the type, but the size of dedicated device being replaced. To accommodate these concerns, ZED will check if the drive being replaced has an allocation bias and then take into account these characteristics in selecting an appropriate spare.

Since a segregated VDEV is allocated from the parent RAIDz or dRAID VDEV, sparing is done in the context of the parent. In other words, sparing the parent will also spare the segregated VDEV.

3.3 Multi-path Support

Lustre servers are deployed in high-availability (HA) pairs in which paired servers have access to each other's storage pools. On failure, the surviving server depends on Linux multi-pathing to mount the other server's storage. As a result, the FMA Diagnosis Engine and Retire Agent must be able to support the Linux architecture. ZFS multi-path support had been started by the ZFS community. We are currently collaborating with the community to ensure their code works with our feature (see this commit for details).

3.4 ZED Watchdog Timer

To prevent a hung zedlet from hanging ZED all together, a watchdog timer (10s) is included to keep zedlets from hanging.

3.5 Multi-Fault Support

A RAIDZ VDEV can handle multiple drive failures in parallel. The structure of the block pointer tree traversal effectively enables queueing of subsequent failures. Reconstruction of a second drive can proceed after the repair of the first driver completes ahead of it in the tree. Because scanning metaslabs during the dRAID sequential rebuild is a serial process, dRAID cannot repair a second driver until the first failure is completely rebuilt. As a result, while rebuilding one failed drive, dRAID does not have the ability to queue subsequent failures.

Due to the way ZED interacts with the ZFS kernel module when it attempts to attach a spare drive to a pool, if a rebuild or resilver is in progress it will be told that the pool is busy and the attach cannot happen. Without multi-fault support, ZED would mark the spare attempt as resolved and move on to processing other events, thus losing the event and need to rebuild the dRAID. The current Retire Agent in ZED was modified to save off the spare request to be replayed later.

The Retire Agent also receives `resilver_finished` and `rebuild_finished` events. When either of these events arrives, the Retire Agent will check for a saved spare request and, if it finds one, will replay the request to begin rebuilding the failed drive. This is implemented so that any number of drives can be faulted and spared-in as long as there are spares available.

4 Metadata Isolation

4.1 Introduction

Metadata Isolation improves large block streaming performance by ensuring that large allocations for application data are not impeded by smaller allocations for ZFS metadata and the subsequent data fragmentation that results when smaller and larger blocks compete for space in ZFS allocation areas (metaslabs).

ZFS already includes the concept of separate storage classes associated with a ZFS pool to create write and read caches (SLOG and L2ARC, respectively) to improve storage performance. The Metadata Isolation feature adds a new allocation class ("special") to hold specific ZFS metadata types and small block application data separate from large block application data.

Allocation classes can be thought of as allocation tiers that are dedicated to specific block categories: the special class, which captures ZFS metadata and small block application data, will occupy a mirrored VDEV and the normal class, which captures all application data, will consume the dRAID or RAIDZ VDEV. Metadata Isolation is an independent feature from the underlying RAID mechanism used for the normal class data.

Exercising the feature requires that each pool be comprised of at least one (special) class. As with the cache or log classes, a special allocation class VDEV can be added at the time the storage pool is created or it can be added at a later time. If the special allocation VDEV is written after the pool is created, only newly written metadata will reside in the newly added VDEV. In general, the capacity of an allocation class VDEV can be expanded by adding additional VDEVs to that class or by replacing existing VDEV devices with a larger device (via ZFS auto-expand).

Each class type represents exclusive allocations but metadata types can be combined onto the associated VDEV. The normal class will accept all block types not being steered into the special class already and serves as the fallback allocator for all classes.

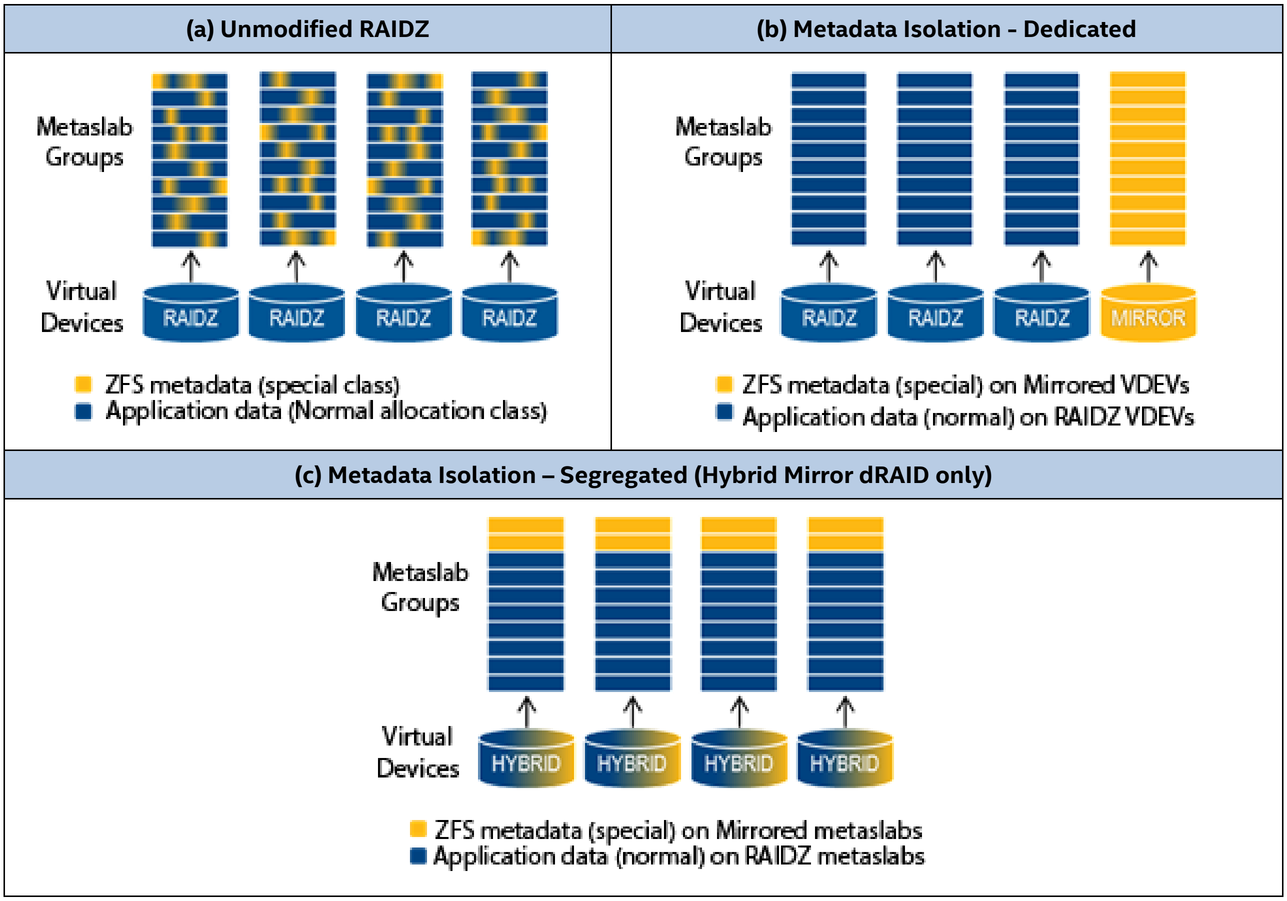
Isolating ZFS metadata and small block I/O to a separate mirrored VDEV decreases fragmentation within the normal class so that allocations of large, contiguous data blocks is less constrained and data can be streamed more efficiently to and from each disk.

The cost of using a dedicated VDEV to isolate the metadata , however, is if the disks in the VDEV are in the same enclosure as the disks used for dRAID, those mirrored disks are unavailable to participate in the dRAID rebuild following a drive failure.

The Metadata Isolation solution created is a hybrid mirror VDEV that combines the mirrored ZFS metadata in separate regions of the dRAID VDEV. Metadata Isolation can be combined on the same disks with dRAID by defining class allocation functionality at the metaslab level and mixing metaslabs in the same VDEV. The Hybrid solution makes all drives available for data and ensures their full participation in the dRAID rebuild following a drive failure.

Metadata Isolation enables different allocation policies for each ZFS data class. Application data can use RAIDZ VDEVs while ZFS metadata and small file system data can use mirrored VDEVs. There are two Opt-in variations for allocation classes: **Dedicated** and **Segregated**.

Figure 4-1. Transition from Unmodified RAIDZ to Hybrid Mirror Configuration



The diagram above (Figure 4-1) illustrates the transition from unmodified RAIDZ configurations to the Hybrid Mirror configuration:

a. Unmodified RAIDz:
   The application data and ZFS metadata are co-allocated within ZFS metaslabs of RAIDZ VDEVs.
b. Metadata Isolation – Dedicated
   The mirrored VDEV for the ZFS metadata and small block I/O (i.e., the "special" allocation class) is created from a separate set of physical VDEVs dedicated for the special class.
c. Metadata Isolation -- Segregated
   The mirrored VDEV is created from a set of metaslabs allocated from a dRAID VDEV. In this way, the metaslabs for the special class are segregated from the metaslabs used for the normal class.

These variations are mutually exclusive use cases. A VDEV may only use one type.

4.2 Dedicated VDEVs

All metaslabs in the VDEV are dedicated to a specific allocation class category. A pool must always have at least one general (non-specified) VDEV when using dedicated VDEVs. This configuration is selected as an “Opt-in” using a VDEV class designation keyword when creating the VDEV. Valid designation keywords are :

```
special | log
```

The dedicated class can currently only be specified with mirrored VDEVs.

Example Syntax:

```
zpool create demo raidz <disks> special mirror <disks>
zpool add demo special mirror <disks>
```

The first command creates the demo pool with ‘special’ class on the specified mirror disks. The second command adds additional disks to the special mirror created by the 1st command. Adding disks is only possible with dedicated VDEVs.

4.3 Segregated VDEVs

In the segregated use case, a portion of a VDEV's metaslabs are set aside for a specific allocation class when the pool is created. Opt-in for this feature is global to the pool using a boolean pool property. The following properties have been defined:

```
segregate_log=on
segregate_special=on
```

These “segregate” properties can be combined if multiple segregation categories are desired (e.g., segregate log class and segregate special class from normal class).

Example Syntax:

```
zpool create -o segregate_special=on demo raidz <disks>
```

This command creates a RAIDz pool named demo with segregation enabled for the special allocation class.

4.3.1 Segregation Percentage

Segregated VDEVs can only be created during zpool creation. It is not possible to add additional segregated VDEVs to an allocation class at a later time, as is possible with dedicated VDEVs.

By default, the following segregation limits are applied:

- segregate_log -- sets aside one metaslab per VDEV for the log class.
- segregate_special -- sets aside 20% of a VDEV's metaslabs for the special class.

The segregated metaslabs are dynamically assigned using a first available algorithm when the VDEV is opened. Subsequent opens may shift which metaslabs are assigned, but once a metaslab is allocated from (i.e. it is activated), the preferred bias becomes persistent.

The percentage allocated to the special class can be tuned to a maximum of 50 percent. Although the number of metaslabs representing the selected percentages is set at pool creation, the assignment of an individual metaslab to the class is deferred until the allocation is needed. Nonetheless, because the ditto block policy (Section 4.3.2) requires writing ditto copies to three different metaslabs, the minimum number of metaslabs initially assigned to the "special" allocation class on a segregated metadata VDEV is three.

ZFS metadata has priority for the special allocations. Since the special class includes allocations for both ZFS metadata and small block data, 5% of a segregated VDEV's metaslabs are reserved for ZFS metadata. This policy will prevent small block usage of the special allocation from competing with ZFS metadata usage of the storage. Small block data can take advantage of the segregated VDEV as long as space is available, otherwise small block data can easily overflow to the normal class.

Block category allocation accounting can be observed from the CLI (see `zdb -mm` and `zpool list -C`).

4.3.2 Ditto Block Policy

Each ZFS block pointer structure has space for the three data virtual address (DVA) pointers. ZFS replicates its metadata and uses the DVAs to record the locations of these "ditto blocks." When there is more than one VDEV, the ditto blocks are written to different VDEVs in the pool. When there is only one VDEV available and more than one DVA is required (ditto copies > 1), the traditional ditto placement policy was to place the allocation a distance of 1/8th of total VDEV allocation space away from the other DVAs. This policy put a burden on the allocator to find a metaslab 1/8th above or below the current allocation.

A new, simplified ditto placement policy has been created to guarantee that the other DVAs simply land in a different metaslab. This policy in turn greatly simplifies ditto DVA placement from a segregated VDEV, where a group of metaslabs is not necessarily consecutive.

To validate that the new policy is honored, a zdb(8) block audit will report any DVAs that landed in the same metaslab. The expected result is that there will be none:

```
#zdb -b ssu_1ost1
Traversing all blocks to verify nothing leaked ...

loading space map for vdev 0 of 1, metaslab 290 of 291 ...
26.4T completed (357720MB/s) estimated time remaining: 0hr 00min 00sec
    No leaks (block sum matches space maps exactly)
```

If there is a policy failure, it will be manifest as a non-zero block audit, as shown in the following zdb audit output in which the ditto block allocation was manipulated to force the error:

```
    Dittoed blocks in same metaslab: 21
```

4.4 VDEV Changes

4.4.1 Feature Flag Encapsulation

The `feature@allocation_classes` becomes active when a unique allocation class is instantiated by a VDEV. Activating this feature makes the pool read-only on ZFS builds that do not support allocation classes.

4.4.2 VDEV Allocation Bias

The ZFS concept of VDEV allocation bias is extended beyond the normal or log classes to include special and segregated classes. Instead of defining a number of Boolean flags, the allocation classes are now expressed in a runtime VDEV instance as an allocation bias:

```
typedef enum vdev_alloc_bias {
    VDEV_BIAS_NONE,
    VDEV_BIAS_LOG,          /* dedicated to ZIL data (SLOG) */
    VDEV_BIAS_DEDUP,        /* dedicated to DDT data */
    VDEV_BIAS_METADATA,     /* dedicated to metadata (DMU and MOS) */
    VDEV_BIAS_SPECIAL,      /* dedicated to small blocks */
    VDEV_BIAS_SEGREGATE,    /* segregated metaslabs into multiple groups */
} vdev_alloc_bias_t;
```

This VDEV allocation class bias information is stored in the per-vdev zap object as a string value:

```
/* vdev metaslab allocation bias */
#define VDEV_ALLOC_BIAS_LOG             "log"
#define VDEV_ALLOC_BIAS_SPECIAL         "special"
#define VDEV_ALLOC_BIAS_SEGREGATE       "segregate"
```

The bias is also passed internally in the pool config during a zpool create and any internal zpool config query. This information can be used by functions in the zpool(8) command.

4.4.3 Metaslab Allocation Bias

Class allocation bias occurs at a metaslab granularity. Each metaslab has an allocation bias which is assigned when the metaslab is initialized, based on the VDEV's allocation bias. The metaslab's allocation bias then determines which metaslab group to join.

There is additional VDEV metadata stored in the VDEV_TOP_ZAP_METASLAB_INFO_OBJ object. This object is an array of ms_alloc_phys structures, one per metaslab, which tracks the allocation bias assigned to a metaslab and the allocation stats by category:

```
/*
 * Additional per-metaslab allocation info for dedicated/segregated vdevs
```

```
 */
typedef struct ms_alloc_phys {
    uint64_t    ms_alloc_flags;         /* flags: ie segregated bias */
    uint64_t    ms_alloc_metadata;      /* metadata space allocated */
    uint64_t    ms_alloc_smallblks;     /* smallblks space allocated */
    uint64_t    ms_alloc_dedup;         /* dedup space allocated */
} ms_alloc_phys_t;
```

Metaslabs in dedicated VDEVs inherit the bias of the VDEV. However, in segregated VDEVs, the class allocation bias of a metaslab is assigned when the metaslab is initialized. The metaslab's allocation bias then determines which metaslab group to join.

```
/*
 * class allocation bias (segregated vdevs only)
 */
typedef enum {
    MS_ALLOC_BIAS_UNASSIGNED =      0x00,
    MS_ALLOC_BIAS_LOG =             0x01,
    MS_ALLOC_BIAS_SPECIAL =         0x02,
    MS_ALLOC_BIAS_NORMAL =          0x03
} ms_alloc_bias_t;
```

4.4.4 VDEV Allocation Stats

Status of zpool VDEVs is available through the zpool list `-v` command, where the mirrored dedicated VDEVs are shown as distinct members of the pool. In the example below, two drives are mirrored to create a dedicated VDEV for the special allocation class. The pool also includes two dRAID virtual spare drives.

```
$ zpool list -v ost-d
NAME                             SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP
ost-d                           16.6T  12.9G  16.6T         -     0%     0%
  draid2                        16.2T  11.2G  16.2T         -     0%  0.06%
    wwn-0x5000c5007adc15a5          -      -      -         -      -      -
    wwn-0x5000c5007adc6d2f          -      -      -         -      -      -
    wwn-0x5000c5007adcf3f4          -      -      -         -      -      -
    wwn-0x5000c5007add017e          -      -      -         -      -      -
    wwn-0x5000c5007addaf56          -      -      -         -      -      -
    wwn-0x5000c5007adc6d4a          -      -      -         -      -      -
    wwn-0x5000c5007b066251          -      -      -         -      -      -
    wwn-0x5000c5007b067415          -      -      -         -      -      -
    wwn-0x5000c5007b065a87          -      -      -         -      -      -
    wwn-0x5000c5007add62b4          -      -      -         -      -      -
    wwn-0x5000c5007addb524          -      -      -         -      -      -
    wwn-0x5000c5007add4c29          -      -      -         -      -      -
    wwn-0x5000c5007add5274          -      -      -         -      -      -
```

```
    wwn-0x5000c5007add5c4b      -      -      -          -      -      -
    wwn-0x5000c5007adc7092      -      -      -          -      -      -
    wwn-0x5000c5007add591d      -      -      -          -      -      -
    wwn-0x5000c5007b34afa6      -      -      -          -      -      -
    wwn-0x5000c5007add870f      -      -      -          -      -      -
    wwn-0x5000c5007b06e13e      -      -      -          -      -      -
    wwn-0x5000c5007b067081      -      -      -          -      -      -
  special:mirror             372G  1.62G   370G          -     0%  0.43%
    wwn-0x55cd2e404c033d2e      -      -      -          -      -      -
    wwn-0x55cd2e404c033fac      -      -      -          -      -      -
spare                           -      -      -          -      -      -
  $draid2-0-s0                  -      -      -          -      -      -
  $draid2-0-s1                  -      -      -          -      -      -
```

Segregated VDEVs, however, are essentially a subset of the main RAID VDEV and, as a result, status of a segregated VDEV is not available through the "`zpool list -v`" command. The special class allocation information is added to the `vdev_stat_t` structure to track the number of metaslabs assigned to the special class and the space used by the normal and special metaslabs that have been assigned.

```
typedef struct vdev_stat {
...
     uint64_t        vs_normal_assigned;     /* ms assigned space    */
     uint64_t        vs_special_assigned;    /* ms assigned space    */
     uint64_t        vs_special_alloc;       /* special allocated    */
} vdev_stat_t;
```

This information is primarily used by the '`zpool list -C`' command to query the allocation info by class category and helps determine if provisioning was done correctly.

```
# zpool list -C ssu_1ost1
NAME            SIZE   ALLOC  FREE   CAPACITY
--------------  -----  -----  -----  --------
ssu_1ost1       72.7T  56.0T  16.8T  77.0%
  draid2-0      72.7T  56.0T  16.8T  77.0%
    normal      58.2T  55.8T  2.48T  95.8%
    special     2.25T   217G  2.04T  9.43%
    unassigned  12.2T      0  12.2T      -
```

This example shows that 2.25TB have been assigned to the special class, but only 217GB have been used. The normal class has allocated 55.8TB of the 58.2TB available from the metaslabs assigned to this class. The pool still has over 20% of its metaslabs (12.2TB) unassigned.

4.5 Notes on Metadata Isolation

- Allocation classes are currently incompatible with PR#5258 (Open ZFS 7090 -- zfs should improve allocation order and throttle allocations). The module parameter `zio_dva_throttle_enabled` is set to B_FALSE in this patch and must remain disabled.
- Enabling the segregated VDEV feature is limited to pool create. The ability to enable it on an existing pool is a stated design goal but requires further testing. It is prevented (by means of a set-once property) as a conservative measure in current builds.
- The effect of causing parity hot spotting by the segregating of metadata away from file data is not known.
- The threshold `zfs_class_smallblk_limit` is a runtime global and should reside in the pool since the metaslab level accounting depends on it not changing.
- At this point there are no custom allocation policies and all classes use the default allocator.

5 Validation

5.1 Building and installing the ZFS Test Suite

The ZFS Test Suite runs under the test-runner framework. This framework is built alongside the standard ZFS utilities and is included as part of zfs-test package. The zfs-test package can be built from source as follows:

```
$ ./configure
$ make pkg-utils
```

The resulting packages can be installed using the rpm or dpkg command as appropriate for your distributions. Alternately, if you have installed ZFS from a distributions repository (not from source) the zfs-test package may be provided for your distribution.

```
- Installed from source
$ rpm -ivh ./zfs-test*.rpm, or
$ dpkg -i ./zfs-test*.deb,

- Installed from package repository
$ yum install zfs-test
$ apt-get install zfs-test
```

5.2 Running the ZFS Test Suite

The pre-requisites for running the ZFS Test Suite are:

- Three scratch disks
  - Specify the disks you wish to use in the $DISKS variable, as a space delimited list like this: DISKS='vdb vdc vdd'. By default the zfs-tests.sh sciprt will construct three loopback devices to be used for testing: DISKS='loop0 loop1 loop2'.
- A non-root user with a full set of basic privileges and the ability to sudo(8) to root without a password to run the test.
- Specify any pools you wish to preserve as a space delimited list in the $KEEP variable. All pools detected at the start of testing are added automatically.
- The ZFS Test Suite will add users and groups to test machine to verify functionality. Therefore it is strongly advised that a dedicated test machine, which can be a VM, be used for testing.

Once the pre-requisites are satisfied simply run the zfs-tests.sh script:

```
$ /usr/share/zfs/zfs-tests.sh
```

Alternately, the zfs-tests.sh script can be run from the source tree to allow developers to rapidly validate their work. In this mode the ZFS utilities and modules from the source tree will be used (rather than those installed on the system). In order to avoid certain types of failures you will need to ensure the ZFS udev rules are installed. This can be done manually or by ensuring some version of ZFS is installed on the system.

```
$ ./scripts/zfs-tests.sh
```

The following zfs-tests.sh options are supported:

Test	Description
-v	Verbose zfs-tests.sh output When specified additional information describing the test environment will be logged prior to invoking test-runner. This includes the runfile being used, the DISKS targeted, pools to keep, etc.
-q	Quiet test-runner output. When specified it is passed to test-runner(1) which causes output to be written to the console only for tests that do not pass and the results summary.
-x	Remove all testpools, dm, lo, and files (unsafe). When specified the script will attempt to remove any leftover configuration from a previous test run. This includes destroying any pools named testpool, unused DM devices, and loopback devices backed by file-VDEVs. This operation can be DANGEROUS because it is possible that the script will mistakenly remove a resource not related to the testing.
-k	Disable cleanup after test failure. When specified the zfs-tests.sh script will not perform any additional cleanup when test-runner exists. This is useful when the results of a specific test need to be preserved for further analysis.
-f	Use sparse files directly instread of loopback devices for the testing. When running in this mode certain tests will be skipped which depend on real block devices.
-d DIR	Create sparse files for VDEVs in the DIR directory. By default these files are created under /var/tmp/.
-s SIZE	Use VDEVs of SIZE (default: 2G)
-r RUNFILE	Run tests in RUNFILE (default: linux.run)

The ZFS Test Suite allows the user to specify a subset of the tests via a runfile. The format of the runfile is explained in test-runner(1), and the files that zfs-tests.sh uses are available for reference under /usr/share/zfs/runfiles. To specify a custom runfile, use the -r option:

```
$ /usr/share/zfs/zfs-tests.sh -r my_tests.run
```

5.3 Test Results

While the ZFS Test Suite is running, one informational line is printed at the end of each test, and a results summary is printed at the end of the run. The results summary includes the location of the complete logs, which is logged in the form /var/tmp/test_results/[ISO 8601 date]. A normal test run launched with the zfs-tests.sh wrapper script will look something like this:

```
$ /usr/share/zfs/zfs-tests.sh -v -d /mnt
```

```
--- Configuration --- Runfile: /usr/share/zfs/runfiles/linux.run STF_TOOLS:
/usr/share/zfs/test-runner STF_SUITE: /usr/share/zfs/zfs-tests FILEDIR: /mnt
FILES: /mnt/file-vdev0 /mnt/file-vdev1 /mnt/file-vdev2 LOOPBACKS: /dev/loop0
/dev/loop1 /dev/loop2 DISKS: loop0 loop1 loop2 NUM_DISKS: 3 FILESIZE: 2G Keep
pool(s): rpool

/usr/share/zfs/test-runner/bin/test-runner.py -c
/usr/share/zfs/runfiles/linux.run -i /usr/share/zfs/zfs-tests Test:
.../tests/functional/acl/posix/setup (run as root) [00:00] [PASS] ...470
additional tests... Test: .../tests/functional/zvol/zvol_cli/cleanup (run as
root) [00:00] [PASS]

Results Summary PASS    472

Running Time:     00:45:09 Percent passed:      100.0% Log directory:
      /var/tmp/test_results/20160316T181651
```

5.4 ZTest/zloop Verification Tests

Ztest and zloop have been modified to test new functionality related to ABD, Allocation Classes and dRAID. When creating configurations, zloop and ztest will randomly opt for creating dRAID pools and opt to turn allocation classes on for those pools. In addition randomly through the tests it will flip linear vs scatter gather allocation on and off for ABD. The Allocation classes and dRAID functionality can be specified through a new set of command line options to both Ztest and Zloop.

Table 5-1. zTest dRAID Options

Parameter	Default	Description
-K<kind>	random	raidz| draid| random
-D <number>	4	Data drives per redundancy group
-G <number>	2	Number of redundancy groups
-S <number>	1	Distributedspare drives
-R<number>	1	RAID group parity
-s <number>	128M	Size of each leaf disk
Example:		
`ztest-VVV -K draid-D 4 -G 3 -S 1 -R 1 -s 256m`		

Appendix A. Usage Examples

A.1 Usage Examples of dRAID for ZFS

A.1.1 Arbitrary Pool Configuration

dRAID supports all RAIDz parity levels. The new "draidcfg" tool is currently required to find the best set of random base permutations for the specified array configuration.

The dRAID pool configuration starts with a new command, "draidcfg," to find the best set of random permutations for the specified input parameters. The output from the draidcfg tool is a configuration file that 'zpool create' uses when creating the dRAID. The list of base permutations in the configuration file will be stored in the ZFS disk labels during zpool creation.

In this example, a triple-parity dRAID is created from 80 drives with seven 8+3 redundancy groups and three distributed spares.

```
# draidcfg -n 80 -d 8 -p 3 -s 3 80.nvl
 Worst ( 7 x 11 + 3) x 5120: 0.998
Seed chosen: b79e65a91d440fc
```

The output from the draidcfg tool indicates the resulting configuration and the random seed that gave the best distribution of the parity groups and distributed spares. The output file holds the list of base permutations.

The following shows the first three base permutations contained in the 80.nvl file created above. The complete file listing is available in Appendix B.3.

```
# draidcfg -r 80.nvl
dRAID3 vdev of 80 child drives: 7 x (8 data + 3 parity) and 3 distributed
spare
Using 64 base permutations
 23,54,38,76,61,14,34,48, 9,31,52,10, 3,41,46,70, 1,
6,59,47,28,32,29,49,30,22,27,11,44,20,56, 5,74,
8,50,15,62,66,33,67,16,65,36,71,75,18,68,21,69,26,64,60,55,42,43,63,35,37,24,
7,17,45, 0, 2,58,78,57,13,12,72,73, 4,19,25,51,79,39,53,77,40,
 41,54,75,48, 2,57,36, 8,76,44, 5, 3,22,30,61,69,47,28,13, 0,
6,71,34,55,33,46,70,79,66,45,27,74,18,25,60,72,11,50,68, 1,53,32,19,64,40,51,
4,31,17,62,42,39,26,56, 7,16,24,12,38,15,78,35,37,67,
9,23,20,49,10,43,14,59,77,29,63,73,58,52,21,65,
 14,65,43, 9,16,53,46,69,17,40,20, 3,47,70,28,39,54, 5,12,24,78,
2,49,61,11,51,75,79,41,50,73,34,18,21,25,52,44,22,32,77, 8,59,15, 7,74,66,
0,71,45,56, 4,36,58,23,68, 6,67,42,29,64,26,33,72,10,37,13,
1,76,60,38,48,31,63,27,35,62,55,19,30,57,
. . . .
```

The command line syntax for creating the dRAID pool is similar to that for creating a RAIDz pool, with the addition of the draidcfg filename before listing the drives to be used in the pool.

```
# zpool create -f -o ashift=12 -o cachefile=none -o segregate_special=on -O
recordsize=16MB MS09 draid3 cfg=/root/80.nvl sdb sdd sde sdg sdh sdi sdk sdl
```

```
sdm sdo sdp sdq sds sdt sdu sdw sdx sdy sdz sdab sdac sdad sdae sdc sdf sdj
sdn sdr sdv sdaa sdaf sdag sdah sdai sdaj sdak sdal sdam sdan sdao sdap sdaq
sdar sdas sdat sdau sdav sdaw sdax sday sdaz sdba sdbb sdbc sdbd sdbe sdbf
sdbg sdbh sdbi sdbj sdbk sdbl sdbm sdbn sdbo sdbp sdbq sdbr sdbs sdbt sdbu
sdbv sdbw sdbx sdby sdbz sdca sdcb sdcd
```

Zpool status for this pool lists all 80 drives contained in the dRAID and the three distributed spare VDEVs. The beginning and end of the listing are shown in the following table. The complete listing is available in Appendix B.4.

```
# zpool status
pool: MS09
 state: ONLINE
 scan: none requested
config:
NAME STATE READ WRITE CKSUM
 MS09 ONLINE 0 0 0
 draid3-0 ONLINE 0 0 0
       sdb ONLINE 0 0 0
       sdd ONLINE 0 0 0
       sde ONLINE 0 0 0

 . . . .

       sdca ONLINE 0 0 0
       sdcb ONLINE 0 0 0
       sdcd ONLINE 0 0 0
 spares
       $draid3-0-s0 AVAIL
       $draid3-0-s1 AVAIL
       $draid3-0-s2 AVAIL
errors: No known data errors
```

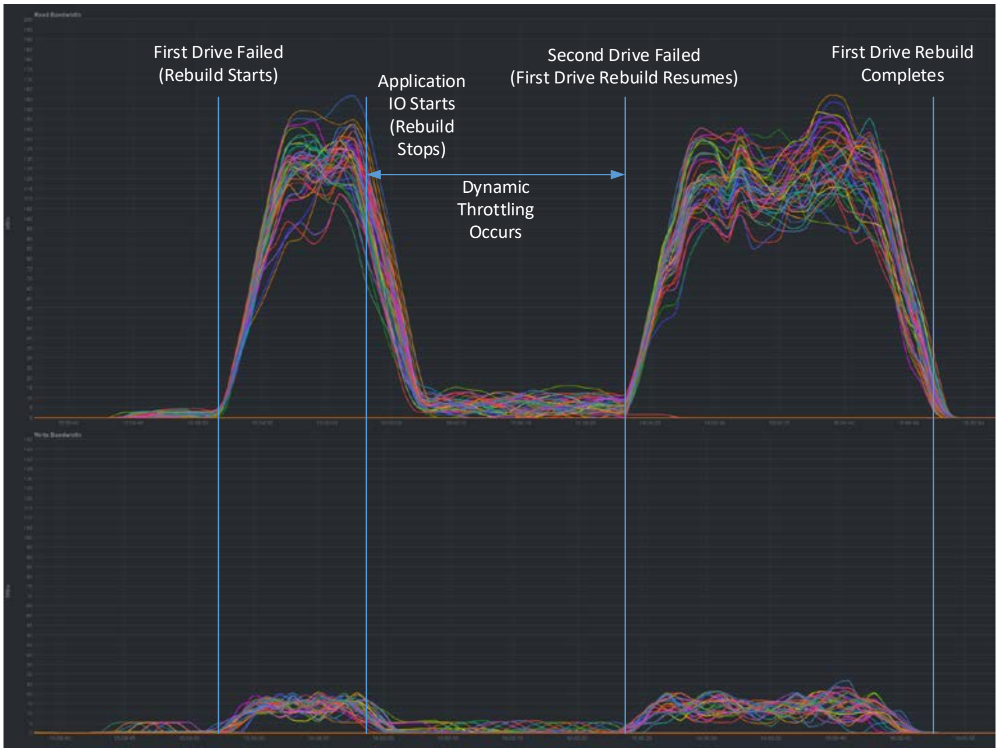
A.1.2 Dynamic Rebuild Throttling

The rebuild process observes and responds to changes in application I/O and pool redundancy level, then throttles itself accordingly. This example shows that the rebuild process :

- slowed down to give more I/O resources to the application
- sped up when the pool lost all redundancy critical mode (2 drives failed on dRAID2) and reached a critical state.

For this test we used a 43 drive Lustre OST. The results of the test are displayed with netdump (Figure 5-1). The upper graph shows read throughput in MB/s of all 43 individual drives. The lower graph shows the write throughput. The Y axis is throughput in MB/s, and the X axis is wallclock time, moving from right to left (older times are on the left).

Figure 5-1. Dynamic Rebuild Throttling



To initiate the test, one drive is taken offline and then started the rebuild by manually replacing the offline drive with a distributed spare.

```
# zpool offline ssu_1ost0 sde
# zpool replace ssu_1ost0 sde '$draid2-0-s0'
```

With no application I/O, the rebuild proceeded at full speed. The md5sum application was then run to generate I/O by reading a list of files from the file system.

```
# md5sum /ssu_1ost0/*.iso
```

As seen on the netdump plot, the rebuild process throttled itself as soon as the application i/O started. Both reads and writes dropped as the drives were busy switching between the two i/O streams from the application and the rebuild. Since this application is not I/O intensive and only did read I/Os, the write I/Os indicate the rebuild is proceeding slowly in the background. The application IO kept the file system busy enough to demonstrate the rebuild throttling mechanism

When another drive is taken offline before the first rebuild completes, the pool reaches a critical state since redundancy of the dRAID2 array has been lost and the rebuild throughput resumes to full speed immediately.

```
# zpool offline ssu_1ost0 sdk
```

When the rebuild completes, the zpool status shows that drive *sde* has been replaced by *$draid2-0-s0*, which is now *INUSE.*

```
# zpool status
NAME STATE READ WRITE CKSUM
ssu_1ost0 DEGRADED 0 0 0
 draid2-0 DEGRADED 0 0 0
  sdb ONLINE 0 0 0
  sdd ONLINE 0 0 0
  spare-2 DEGRADED 0 0 0
   sde OFFLINE 0 0 0
   $draid2-0-s0 ONLINE 0 0 0
  sdg ONLINE 0 0 0
  sdh ONLINE 0 0 0
  sdi ONLINE 0 0 0
  sdk OFFLINE 0 0 0
......
spares
 $draid2-0-s0 INUSE currently in use
 $draid2-0-s1 AVAIL
$draid2-0-s2 AVAIL
```

A.1.3 Rebuild Stop and Resume

Rebuild progress is periodically persisted so that if the rebuild process t is interrupted, the rebuild will be able to resume again from where progress was last saved, rather than restarting from the beginning.

In the previous example, two drives were taken offline (*sde* and *sdk*) and rebuilt *sde* onto *$draid2-0-s0*. For this example, the rebuild of *sdk* was begun onto *$draid2-0-s1.* The rebuild was then interrupted by exporting the pool.

```
# zpool replace ssu_1ost0 sdk '$draid2-0-s1'
# zpool export ssu_1ost0
```

The Linux dmesg log is used to view the debug messages from the rebuild process showing that rebuild was interrupted at metaslab 31.

```
[265158.886650] Scanning 4 segments for MS 30
[265158.892179] MS (30 at 8053063680K) segment: 720K + 80K
[265158.898894] MS (30 at 8053063680K) segment: 1920K + 80K
[265158.905767] MS (30 at 8053063680K) segment: 2080K + 80K
[265158.912586] MS (30 at 8053063680K) segment: 2880K + 80K
[265158.919333] Completed rebuilding metaslab 30
[265158.924989] All metaslabs [0, 29) fully rebuilt.
[265158.931215] Scanning 4 segments for MS 31
[265158.936432] MS (31 at 8321499160K) segment: 160K + 81920K
```

```
[265158.943346] MS (31 at 8321499160K) segment: 98280K + 101785600K
[265160.549318] Completed rebuilding metaslab 29
[265160.555119] All metaslabs [0, 31) fully rebuilt.
[265163.080670] Aborted rebuilding metaslab 31
```

Note that the rebuild progress shown in the debug message log represents the saved status in-memory. The on-disk persisted progress usually lags behind the saved in-memory state by a number of metaslabs. As a result, the rebuild is expected to resume metaslab 31 or earlier.

When the pool is imported, rebuild resumed from the progress persisted to disk. The following debug messages show that the rebuild started from metaslab 29.

```
[265196.221793] Restarting rebuild at metaslab 29
[265197.190622] Scanning 36 segments for MS 29
[265197.195950] MS (29 at 7784628240K) segment: 0K + 229120K
[265197.208742] MS (29 at 7784628240K) segment: 229360K + 491480K
[265197.220629] MS (29 at 7784628240K) segment: 720880K + 264320K
[265197.227520] MS (29 at 7784628240K) segment: 985360K + 106320K
[265197.234404] MS (29 at 7784628240K) segment: 1091760K + 49960K
[265197.241346] MS (29 at 7784628240K) segment: 1141800K + 37960K
[265197.248362] MS (29 at 7784628240K) segment: 1179840K + 37200K
```

After the rebuild is completed, both distributed spares are now *INUSE*:

```
# zpool status
NAME STATE READ WRITE CKSUM
ssu_1ost0 DEGRADED 0 0 0
 draid2-0 DEGRADED 0 0 0
  sdb ONLINE 0 0 0
  sdd ONLINE 0 0 0
  spare-2 DEGRADED 0 0 0
   sde OFFLINE 0 0 0
   $draid2-0-s0 ONLINE 0 0 0
  sdg ONLINE 0 0 0
  sdh ONLINE 0 0 0
  sdi ONLINE 0 0 0
  spare-6 DEGRADED 0 0 0
   sdk OFFLINE 0 0 0
   $draid2-0-s1 ONLINE 0 0 0
  sdl ONLINE 0 0 0
  sdm ONLINE 0 0 0
......
 spares
  $draid2-0-s0 INUSE currently in use
  $draid2-0-s1 INUSE currently in use
  $draid2-0-s2 AVAIL
```

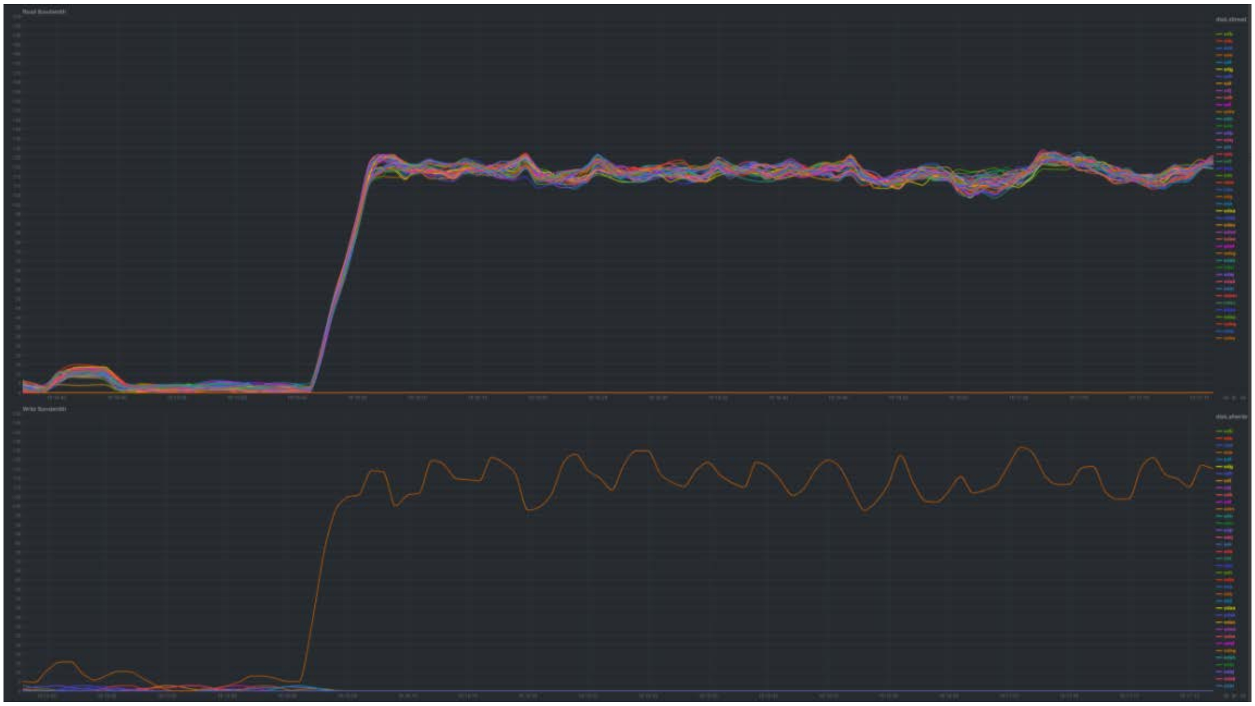
A.1.4 Rebalance

In this example, the failed sde drive is replaced with the sdas drive to make the *$draid2-0-s0* spare available again:

```
# zpool replace -f ssu_1ost0 sde sdas
```

The rebalance process uses the traditional ZFS resilver mechanism. Although it is essentially reconstructing the same lost redundancy as the previous rebuild, rebalance is much slower as it has to traverse the block pointer tree and write to a single spare drive. As shown below (Figure 5-2), only the replacement drive (sdas) does write IO, while during a sequential rebuild all surviving drives share the write workload (Figure 5-1).

Figure 5-2. Rebalance to a Replacement Drive



After rebalance is completed, zpool status reports the corresponding distributed spare (*$draid2-0-s0*) as being available (AVAIL).

```
# zpool status
NAME STATE READ WRITE CKSUM
ssu_1ost0 DEGRADED 0 0 0
 draid2-0 DEGRADED 0 0 0
  sdb ONLINE 0 0 0
  sdd ONLINE 0 0 0
```

```
  sdas ONLINE 0 0 0
  sdg ONLINE 0 0 0
  sdh ONLINE 0 0 0
  sdi ONLINE 0 0 0
  spare-6 DEGRADED 0 0 0
   sdk OFFLINE 0 0 0
   $draid2-0-s1 ONLINE 0 0 0
  sdl ONLINE 0 0 0
  .......
spares
 $draid2-0-s0 AVAIL
 $draid2-0-s1 INUSE currently in use
 $draid2-0-s2 AVAIL
```

A.2 Usage Examples of Metadata Isolation with Lustre* and dRAID

These examples demonstrate the aspects of different allocation class configurations using zpool(8), zdb(8) and kstat.

A.2.1 Hybrid Metadata/Smallblock Isolation with dRAID VDEVs

This example compares two dRAID zpools:

- ssu_1ost1 had VDEV segregation enabled to create a hybrid-mirror dRAID. This dRAID VDEV set aside a portion of its allocation areas (metaslabs) to host metadata and small blocks. The remaining areas were used for generic application data and are intended to stream larger 16MB block content.
- ssu_2ost0 had dRAID alone. In this configuration of dRAID, allocations were not differentiated by category. Each metaslab hosted data as it arrived, which could be any mixture of small or large data and ZFS metadata.

Both dRAID pools had 43 drives in four 8+2 parity groups and 3 spares. Note that the zpool status for both pools shows the three virtual spares “$draid-xx”, but otherwise there is nothing to indicate that ssu_1ost1 also had a hybrid mirror for the special allocation class.

dRAID with VDEV segregation enabled:

```
# zpool status
  pool: ssu_1ost1
 state: ONLINE
  scan: none requested
config:

  NAME              STATE     READ WRITE CKSUM
  ssu_1ost1         ONLINE       0     0     0
    draid2-0        ONLINE       0     0     0
      mpathfn       ONLINE       0     0     0
      mpathfa       ONLINE       0     0     0
      mpathcx       ONLINE       0     0     0
      mpathbs       ONLINE       0     0     0
      mpathan       ONLINE       0     0     0
      mpathu        ONLINE       0     0     0
      mpatheu       ONLINE       0     0     0
      mpathdp       ONLINE       0     0     0
      mpathck       ONLINE       0     0     0
      mpathbf       ONLINE       0     0     0
      mpathaa       ONLINE       0     0     0
      mpathh        ONLINE       0     0     0
      mpatheh       ONLINE       0     0     0
      mpathdc       ONLINE       0     0     0
      mpathfm       ONLINE       0     0     0
      mpathaz       ONLINE       0     0     0
      mpathcw       ONLINE       0     0     0
      mpathbr       ONLINE       0     0     0
      mpatham       ONLINE       0     0     0
      mpatht        ONLINE       0     0     0
      mpathdd       ONLINE       0     0     0
      mpathdo       ONLINE       0     0     0
      mpathcj       ONLINE       0     0     0
      mpathbe       ONLINE       0     0     0
      mpathg        ONLINE       0     0     0
      mpathfl       ONLINE       0     0     0
      mpatheg       ONLINE       0     0     0
      mpathdb       ONLINE       0     0     0
      mpathay       ONLINE       0     0     0
      mpathcv       ONLINE       0     0     0
      mpathbq       ONLINE       0     0     0
      mpathal       ONLINE       0     0     0
      mpaths        ONLINE       0     0     0
      mpathfx       ONLINE       0     0     0
      mpathes       ONLINE       0     0     0
      mpathdn       ONLINE       0     0     0
      mpathci       ONLINE       0     0     0
      mpathbd       ONLINE       0     0     0
      mpathf        ONLINE       0     0     0
      mpathfk       ONLINE       0     0     0
      mpathef       ONLINE       0     0     0
      mpathda       ONLINE       0     0     0
      mpathax       ONLINE       0     0     0
  spares
    $draid2-0-s0  AVAIL
```

dRAID with no metadata isolation:

```
# zpool status
  pool: ssu_2ost0
 state: ONLINE
  scan: none requested
config:

  NAME              STATE     READ WRITE CKSUM
  ssu_2ost0         ONLINE       0     0     0
    draid2-0        ONLINE       0     0     0
      mpathdd       ONLINE       0     0     0
      mpathfa       ONLINE       0     0     0
      mpathcx       ONLINE       0     0     0
      mpathbs       ONLINE       0     0     0
      mpathan       ONLINE       0     0     0
      mpathu        ONLINE       0     0     0
      mpatheu       ONLINE       0     0     0
      mpathdp       ONLINE       0     0     0
      mpathck       ONLINE       0     0     0
      mpathbf       ONLINE       0     0     0
      mpathaa       ONLINE       0     0     0
      mpathh        ONLINE       0     0     0
      mpathfm       ONLINE       0     0     0
      mpatheh       ONLINE       0     0     0
      mpathdc       ONLINE       0     0     0
      mpathaz       ONLINE       0     0     0
      mpathcw       ONLINE       0     0     0
      mpathbr       ONLINE       0     0     0
      mpatham       ONLINE       0     0     0
      mpatht        ONLINE       0     0     0
      mpathet       ONLINE       0     0     0
      mpathdo       ONLINE       0     0     0
      mpathcj       ONLINE       0     0     0
      mpathbe       ONLINE       0     0     0
      mpathg        ONLINE       0     0     0
      mpathfl       ONLINE       0     0     0
      mpatheg       ONLINE       0     0     0
      mpathdb       ONLINE       0     0     0
      mpathay       ONLINE       0     0     0
      mpathcv       ONLINE       0     0     0
      mpathbq       ONLINE       0     0     0
      mpathal       ONLINE       0     0     0
      mpaths        ONLINE       0     0     0
      mpathfx       ONLINE       0     0     0
      mpathes       ONLINE       0     0     0
      mpathdn       ONLINE       0     0     0
      mpathci       ONLINE       0     0     0
      mpathbd       ONLINE       0     0     0
      mpathfk       ONLINE       0     0     0
      mpathef       ONLINE       0     0     0
      mpathda       ONLINE       0     0     0
      mpathax       ONLINE       0     0     0
      mpathei       ONLINE       0     0     0
  spares
    $draid2-0-s0  AVAIL
```

`    $draid2-0-s1  AVAIL`
`    $draid2-0-s2  AVAIL`
 	
`errors: No known data errors`
	`    $draid2-0-s1  AVAIL`
	`    $draid2-0-s2  AVAIL`
	
	`errors: No known data errors`

The allocation data for the dRAID with segregation enabled can be seen with the '`zpool list –C`'.

The Special Class used in these examples comes from enabling segregation. For dRAID, this is an automatic opt-in as it makes sense to join the two features. This opt-in can further be observed by examining the following pool properties using 'zpool get'. These properties are automatically set with dRAID and are read-only.

```
# zpool get feature@allocation_classes,segregate_special,smallblkceiling
NAME       PROPERTY                     VALUE                      SOURCE
ssu_1ost1  feature@allocation_classes  active                     local
ssu_1ost1  segregate_special           on                         local
```

Using the ZFS kstat framework, one can track the allocations occurring in each of the pool allocation classes while the file system is running.

```
cat /proc/spl/kstat/zfs/alloc_class_stats

name                            type data
normal_allocated                4    61325031915520
normal_highest_allocated        4    61325037486080
special_allocated               4    233487310848
special_highest_allocated       4    233536917504
slog_allocated                  4    0
slog_highest_allocated          4    0
```

A.2.2 Observing Metaslab Regions

Using the zdb(8) tool, one can observe the underlying metaslabs in a zpool. With VDEV segregation enabled, ZFS will set aside a portion (20% by default) of these regions to service small blocks and metadata. The remaining regions are used for generic application data and large streaming I/O.

```
The 'zdb –m' command provides copious output. The zpools created for this
demonstration each had 290 metaslabs. The listing for a pool with dRAID alone
(ssu_2ost0) is shown in Appendix B.1.  A fragment of this listing is below
(the size column has been deleted to make the data
# zdb -m ssu_2ost0
Metaslabs:
    vdev          0
```

```
metaslabs   291   offset                 spacemap          free
---------------   -------------------   ---------------   ------------
metaslab      0   offset             0   spacemap    114   free   7.36M
metaslab      1   offset    4000006000   spacemap    113   free   1.64G
metaslab      2   offset    8000002000   spacemap    112   free    861M
metaslab      3   offset    c000008000   spacemap    123   free   1.04G
metaslab      4   offset   10000004000   spacemap    122   free   1.07G
```

The listing for the dRAID pool segregation enabled is show in Appendix B.2. With segregation enabled, the listing now includes an additional column for class. There are three entries possible :

- ‘special’ means the metaslab is assigned to the special allocation class. This metaslab is part of a mirrored VDEV that contains ZFS metadata and/or small block data.
- ‘normal’ means the metaslab is assigned to the normal class. This metaslab is part of the dRAID VDEV and contains large block application data.
- ‘----‘ means the metaslab is unassigned. It can be assigned to the ‘special’ or ‘normal’ class as soon as ZFS needs an allocation for that class.

```
# zdb -m ssu_lost1
Metaslabs:
    vdev          0   segregate
    metaslabs   291   offset                spacemap          free          class
    ---------------   -------------------   ---------------   ------------  --------
    metaslab      0   offset            0   spacemap    115   free    122G   special
    metaslab      1   offset   4000000000   spacemap    114   free    208G   special
    metaslab      2   offset   8000001000   spacemap    113   free    221G   special
    metaslab      3   offset   c000001000   spacemap      4   free    256G   special
    metaslab      4   offset  10000002000   spacemap      3   free    256G   special
    metaslab      5   offset  14000000000   spacemap      2   free    256G   special
    metaslab      6   offset  18000000000   spacemap      7   free    256G   special
    metaslab      7   offset  1c000001000   spacemap      6   free    256G   special
    metaslab      8   offset  20000001000   spacemap      5   free    256G   special
    metaslab      9   offset  24000002000   spacemap      0   free    256G    ----
    metaslab     10   offset  28000000000   spacemap      0   free    256G    ----
    metaslab     11   offset  2c000000000   spacemap      0   free    256G    ----
    metaslab     12   offset  30000001000   spacemap      0   free    256G    ----
    metaslab     13   offset  34000001000   spacemap      0   free    256G    ----

    . . . .

    metaslab     58   offset  e8000008000   spacemap    123   free   7.70G    normal
    metaslab     59   offset  ec000004000   spacemap    125   free   1.43G    normal
    metaslab     60   offset  f0000000000   spacemap    124   free   1.58G    normal
    metaslab     61   offset  f4000006000   spacemap    126   free   1.27G    normal
    metaslab     62   offset  f8000002000   spacemap    127   free   1.66G    normal
    metaslab     63   offset  fc000008000   spacemap    128   free   2.05G    normal
    metaslab     64   offset 100000004000   spacemap    129   free   2.23G    normal
    metaslab     65   offset 104000000000   spacemap    130   free   2.01G    normal
    metaslab     66   offset 108000006000   spacemap    131   free   1.59G    normal

    . . . .

    metaslab    284   offset 470000004000   spacemap    349   free   12.7G    normal
    metaslab    285   offset 474000000000   spacemap    350   free   21.9G    normal
    metaslab    286   offset 478000006000   spacemap    351   free   19.0G    normal
    metaslab    287   offset 47c000002000   spacemap    352   free   1.16G    normal
    metaslab    288   offset 480000008000   spacemap    353   free    912M    normal
    metaslab    289   offset 484000004000   spacemap    354   free    902M    normal
    metaslab    290   offset 488000000000   spacemap    355   free   1.35G    normal
```

A.2.3 Observing Free Space Fragmentation

We can observe the free space fragmentation details of each metaslab by running '`zdb -mm`' to dump histograms of data allocations in each metaslab. The free space fragmentation affects the new data block allocations and the resulting I/O performance of new files.

In the samples below, the fragmentation histograms are the free segments for a power-of-two size. So 2^13 represents 8KB free chunks and 2^24 represents 16MB free chunks. After an aging run, there typically are no free regions in the non-segregated pool large enough to satisfy a 16MB block on the pool with segregation disabled. In that case, ZFS would have to stitch together a set of blocks to satisfy a 16MB block request.

As expected, the pool with large and small block isolation provided by segregation has different fragmentation characteristics. For a metaslab that is servicing small blocks and metadata, it is acceptable to have lots of smaller blocks that are free since later small allocations can fill in those holes. For a metaslab that is servicing larger blocks, it would ideally have plenty of larger contiguous ares from which to draw from. In the segregated pool, there are still 106+2*9+4*1=128 16MB chunks free in the normal class.

```
metaslab 34  offset 88000004000  size 3fffffc000  spacemap 153  free 21.1G
On-disk histogram:              fragmentation 21
                 15:  31492 ****************************************
                 16:   9869 *************
                 17:   2356 ***
                 18:   1665 ***
                 19:   2275 ***
                 20:   3543 *****
                 21:   2593 ****
                 22:   1097 **
```

As expected, the pool with large and small block isolation provided by segregation has different fragmentation characteristics. For a metaslab that is servicing small blocks and metadata (special class), it is acceptable to have lots of smaller blocks that are free since later small allocations can fill in those holes.

```
metaslab  2  offset 8000001000  size 3ffffff000  spacemap 113  free 221G   special
On-disk histogram:              fragmentation 8
                 13: 448909 ****************************************
                 14: 161708 ***************
                 15:  91420 *********
                 16:  29406 ***
                 17:  13000 **
                 18:   6912 *
                 19:   5800 *
                 20:   2823 *
                 21:   2205 *
                 22:   1317 *
                 23:    819 *
                 24:    431 *
                 25:    211 *
                 26:    163 *
                 27:    135 *
                 28:      2 *
                 29:      0
                 30:      0
                 31:      0
                 32:      0
                 33:      0
                 34:      0
                 35:      0
                 36:      1 *
```

For a metaslab that is servicing larger blocks (normal class), it would ideally have plenty of larger contiguous ares from which to draw from. In the segregated pool, there are still 106+2*9+4*1=128 16MB chunks free in the normal class.

```
metaslab 77  offset 134000002000  size 3fffffe000  spacemap 142  free 19.9G  normal
On-disk histogram:              fragmentation 13
                 16:    285 ***
                 17:    484 *****
                 18:    810 ********
                 19:   1449 **************
                 20:   4302 ****************************************
                 21:   1997 *******************
                 22:    839 ********
                 23:    107 *
                 24:    106 *
                 25:      9 *
                 26:      1 *
```

A.2.4 Observing Allocations by Category

The ‘`zdb -mm`’ command also includes an Allocation Summary section that shows what allocations were made by category. This can be used to confirm that the metaslab regions are allocating from the expected class. Both examples below are from a dump of a zpool with segregation enabled.

For a normal class metaslab, the Allocation Summary shows that all of the blocks allocated to the metaslab are in the generic category.

```
metaslab 75  offset 12c000000000  size 4000000000  spacemap 141  free 24.1G  normal
Allocation Summary:           232G allocated
           metadata:   0.0%
          smallblks:   0.0%
              dedup:   0.0%
            generic: 100.0%  ********************************
```

For a special class metaslab, the blocks allocated belong to the metadata and small block categories.

```
metaslab  0  offset          0   size 4000000000  spacemap 115  free 122G  special
Allocation Summary:           134G allocated
           metadata:  62.1%  ********************
          smallblks:  37.9%  ************
              dedup:   0.0%
            generic:   0.0%
```

A normal class allocation may include metadata and small block categories as well as generic. A special class allocation will only hold metadata and small blocks. The special class cannot hold a generic category allocation.

A.3 Usage Examples of End-to-End 16MB File Block I/Os

The example consists of two parts: End-to-End Streaming, to show the transfer of 16MB from Lustre clients to the ZFS, and Fragmentation Improvements, to show the improved performance with Metadata Isolation. For both examples, we used a cluster that had 8 Lustre clients and 4 Lustre OSSs. Each OSS had a single 43 drive dRAID2 OS.

A.3.1 Configuring the file system for 16MB I/Os

Each file system component along the I/O path must be configured to enable 16MB I/O. Starting from the Lustre server, 16MB I/Os are set first at ZFS, then Linux, then Lustre OSS, and then, finally, the Lustre client.

A.3.1.1 ZFS on the Lustre OSS

On each Lustre OSS, set ZFS to accept and use 16MB I/Os with the following steps:

1. Set "`zfs_max_recordsize`" to 16MB (16777216).

```
# echo "16777216" > /sys/module/zfs/parameters/zfs_max_recordsize
```

2. Create the zpool while specifying a 16MB record size using the "recordsize" option.

```
# zpool create -o ashift=12 -o segregate_special=on -o cachefile=none -O
recordsize=16MB ssu_1ost1 draid2 cfg=test_2_8_3_43_draidcfg.nvl mpathfn
mpathfa mpathcx mpathbs mpathan mpathu mpatheu mpathdp mpathck mpathbf
mpathaa mpathh mpatheh mpathdc mpathfm mpathaz mpathcw mpathbr mpatham mpatht
mpathdd mpathdo mpathcj mpathbe mpathg mpathfl mpatheg mpathdb mpathay
mpathcv mpathbq mpathal mpaths mpathfx mpathes mpathdn mpathci mpathbd mpathf
mpathfk mpathef mpathda mpathax
```

The "-o ashift=12" option is only necessary to force 4KB sectors on hard drives that pretend to have 512-byte sectors for backward compatibility.

3. Enable the ZFS feature@large_blocks flag for the zpool. Verify the feature with the zpool get all command.

```
# zpool feature@large_blocks=enabled ssu_1ost1

# zpool get all ssu_1ost1 |grep large_blocks
ssu_1ost1  feature@large_blocks   active                    local
```

A.3.1.2 Linux on the Lustre OSS

The Linux block I/O layer for each disk drive on the Lustre OSS must be configured to handle 2MB I/Os. This is done by setting the max_sectors_kb parameter to 4096 (512B/sector * 4096 sectors = 2MB) and the scheduler to noop. The following script was run before the example started:

```
for i in $(find /sys/devices -print |grep max_sectors_kb |grep -v ata)
do
  echo 4096 > $i
done


for x in $(find /sys/devices -print |grep scheduler |grep -v ata)
do
  echo noop > $x
done
```

A.3.2 Lustre OSS

The Lustre OSS itself is configured to use 16MB by using the Lustre control interface, lctl, to set the obdfilter read and write size (brw_size) to 16:

```
# lctl set_param obdfilter.*.brw_size=16
```

```
obdfilter.nlsdraid-OST0001.brw_size=16
```

A.3.2.1 Lustre Client

The RPC size used by the Lustre client is controlled by the `max_pages_per_rpc` parameter. Each page is 4096 bytes. By default, Lustre sets `max_pages_per_rpc` to 256 to use 1MB RPCs (256*4096=1048576). Starting in Lustre 2.9, it is possible to raise the parameter to 4096 to use 16MB RPCs (4096*4096 = 16MB). To make this change, we use pdsh to run lctl on each compute node to set the RPC size on the Lustre client for each OSS connection.

```
# pdsh -w node0[1-8] "/usr/sbin/lctl set_param osc.*.max_pages_per_rpc=16M"
node01: osc.nlsdraid-OST0000-osc-ffff8820228e3000.max_pages_per_rpc=4096
node01: osc.nlsdraid-OST0001-osc-ffff8820228e3000.max_pages_per_rpc=4096
node01: osc.nlsdraid-OST0002-osc-ffff8820228e3000.max_pages_per_rpc=4096
node01: osc.nlsdraid-OST0003-osc-ffff8820228e3000.max_pages_per_rpc=4096
...
node04: osc.nlsdraid-OST0000-osc-ffff8816686ea000.max_pages_per_rpc=4096
node04: osc.nlsdraid-OST0001-osc-ffff8816686ea000.max_pages_per_rpc=4096
node04: osc.nlsdraid-OST0002-osc-ffff8816686ea000.max_pages_per_rpc=4096
node04: osc.nlsdraid-OST0003-osc-ffff8816686ea000.max_pages_per_rpc=4096
```

A.3.3 Prepping Lustre Counters

Lustre maintains a number of useful counters on the client and server to help evaluate the performance of different components of the file system. For the End-to-End 16MB demonstration, we used the “rpc_stats” structures on the client and “brw_stats” structure on the server. The scripts used during the demonstration are described below.

A.3.3.1 RPC Stats

RPC stats are kept on the Lustre client to show the distribution of RPCs issued by the client to the Lustre server. The rpc_stats variable on each client can be cleared before the test and the read after the test completes.

A.3.3.2 clear_rpc.sh

The clear_rpc.sh script cleared the rpc_stats counter structure on all 8 clients on the cluster. This script is run before each performance test.

```
#!/bin/bash

for host in node01 node02 node03 node04 node05 node06 node07 node08
do

  echo "clear on $host"
```

```
  ssh $host "/usr/sbin/lctl set_param osc.*.rpc_stats=0"
  echo

done
```

When complete, the script shows that the counters have been zeroed:

```
# ./clear_rpc.sh
clear on node01
osc.nlsdraid-OST0000-osc-ffff8820228e3000.rpc_stats=0
osc.nlsdraid-OST0001-osc-ffff8820228e3000.rpc_stats=0
osc.nlsdraid-OST0002-osc-ffff8820228e3000.rpc_stats=0
```

A.3.3.3 show_rpc.sh

The show_rpc script displays the rpc_stats structure from each Lustre client.

```
#!/bin/bash

for host in node01 node02 node03 node04 node05 node06 node07 node08
do

  ssh $host "/bin/hostname; cat /proc/fs/lustre/osc/nlsdraid-OST000[1-
3]*/rpc_stats | grep -A14 'pages per' " |egrep --color=always '.*4096:.*|$'
  echo

done
```

The output from this script is used to evaluate how the Lustre clients sent RPCs to the servers.

A.3.4 BRW Stats

The Lustre server maintains counters for the block I/O requests that it sends to the underlying Linux file system. The counters are cleared before a test, and then read afterwards.

A.3.4.1 clear_brw.sh

The cluster had 4 Lustre OSS. This script cleared the brw_stats structure on each server.

```
#!/bin/bash

for host in lustre1 lustre2 lustre3 lustre4
done
```

When complete, the script shows that the servers have been zeroed:

```
# ./clear_brw.sh
obdfilter.nlsdraid-OST0000.brw_stats=0
obdfilter.nlsdraid-OST0001.brw_stats=0
obdfilter.nlsdraid-OST0002.brw_stats=0
obdfilter.nlsdraid-OST0003.brw_stats=0
```

A.3.4.2 show_brw.sh

The `show_brw` script shows the block distribution sent to the underlying storage.

```
#!/bin/bash

for host in lustre1 lustre2 lustre3 lustre4
do

  ssh $host "/bin/hostname; cat /proc/fs/lustre/osd-zfs/*/brw_stats|grep -A36
'size' " |egrep --color=always '.*16M.*|$'
  echo

done
```

The output from this script is shown in Section A.4.2.

A.4 End to End Streaming

IOR was used to generate 16MB I/O from the clients using file per process with a sequential workload.

```
# mpirun -wdir /mnt/lustre -np 8 -machinefile hosts /root/natasha-bin/ior -F
-i 1 -s 20480 -b 16m -t 16m
...
Command line used: /root/natasha-bin/ior -F -i 1 -s 20480 -b 16m -t 16m
Machine: Linux node01

Test 0 started: Tue Jun 20 14:37:07 2017
Summary:
    api                = POSIX
    test filename      = testFile
    access             = file-per-process
    ordering in a file = sequential offsets
    ordering inter file= no tasks offsets
    clients            = 8 (1 per node)
    repetitions        = 1
    xfersize           = 16 MiB
    blocksize          = 16 MiB
```

```
    aggregate filesize = 2560 GiB
```

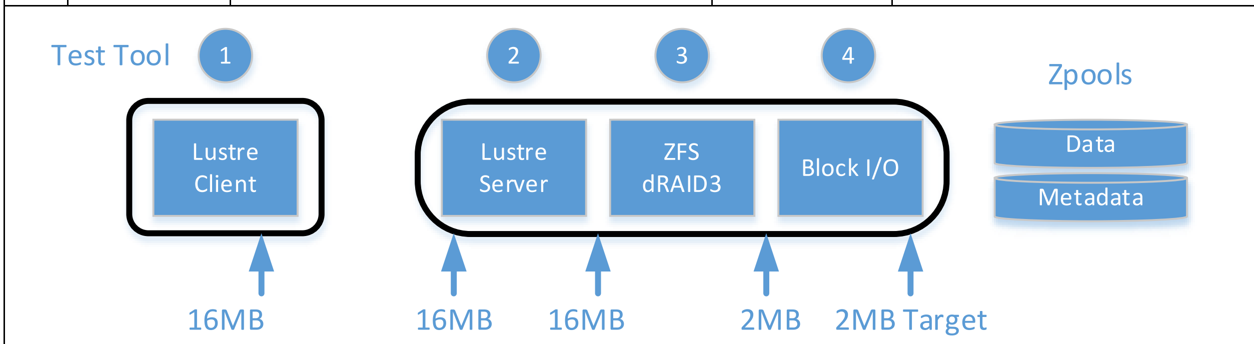
The IOR test was configured so that each of the four OSS received I/O from two different clients:

```
node01 → testFile.00000000 on ost0
node02 → testFile.00000001 on ost1
node03 → testFile.00000002 on ost2
node04 → testFile.00000003 on ost3
node05 → testFile.00000004 on ost0
node05 → testFile.00000005 on ost1
node07 → testFile.00000006 on ost2
node08 → testFile.00000007 on ost3
```

While the workload tests ran, a number of Linux tools were used to expose the I/O sizes at each step of the I/O flow, from the Lustre clients to the ZFS disk devices. These tools are summarized in Table 5-2 and show at what point each tool is used in the I/O flow.

Table 5-2. I/O Size Evaluation Tools

	Metric	Tool	Run on	Display
1	RPC Stats	`Lctl get_param osc.*.rpc_stats`	Lustre Client	Histogram of the RPC sizes and number from the client
2	BRW Stats	`Lctl get_param obdfilter.*.brw_stats`	Lustre Server (OST)	Histograms of RPC sizes received on each OST
3	ZFS iostats	`Zpool iostat -r ost0`	Lustre Server (ZFS)	Tables of request sizes in each zpool issued to disk
4	Disk Stats	`visualized with netdump`	Lustre Server (Linux)	graphic display of



A.4.1 Lustre Client RPC stats

Before running the IOR test, the clear_rpc.sh script (section A.3.3.2) cleared the rcp_stats structure on each Lustre client and the clear_brw.sh script (section A.3.4.1) cleared the brw_stats structure on each Lustre OSS.

The show_rpc.sh script (section A.3.3.3) displayed the rpc_stats structure from each client. With 4KB sized pages, 4096 pages per RPC represent 16MB per RPC. All clients showed a result similar to the one below in which all RPCs sent during the IOR run work 16MB in size.

```
node01 ost0
pages per rpc       rpcs   % cum % |        rpcs   % cum %
1:                     0   0   0   |           0   0   0
2:                     0   0   0   |           0   0   0
4:                     0   0   0   |           0   0   0
8:                     0   0   0   |           0   0   0
16:                    0   0   0   |           0   0   0
32:                    0   0   0   |           0   0   0
64:                    0   0   0   |           0   0   0
128:                   0   0   0   |           0   0   0
256:                   0   0   0   |           0   0   0
512:                   0   0   0   |           0   0   0
1024:                  0   0   0   |           0   0   0
208:                   0   0   0   |           0   0   0
4096:                  0   0   0   |        3500 100 100
```

A.4.2 Lustre Server BRW stats

We then verified that 16MB IO blocks were sent through the Lustre server with the show_brw.sh script (section A.3.4.2):

```
# ./show_brw.sh
ssu1_oss1
disk I/O size           ios    % cum % |    ios    %   cum %
16M:                      0    0    0  |   2048 100   100

ssu1_oss2
disk I/O size           ios    % cum % |    ios     %   cum %
16M:                      0    0    0  |   2048   100   100

ssu2_oss1
disk I/O size           ios    % cum % |    ios     %   cum %
16M:                      0    0    0  | 314290   100   100

ssu2_oss2
disk I/O size           ios    % cum % |    ios     %   cum %
16M:                      0    0    0  | 306408   100   100
```

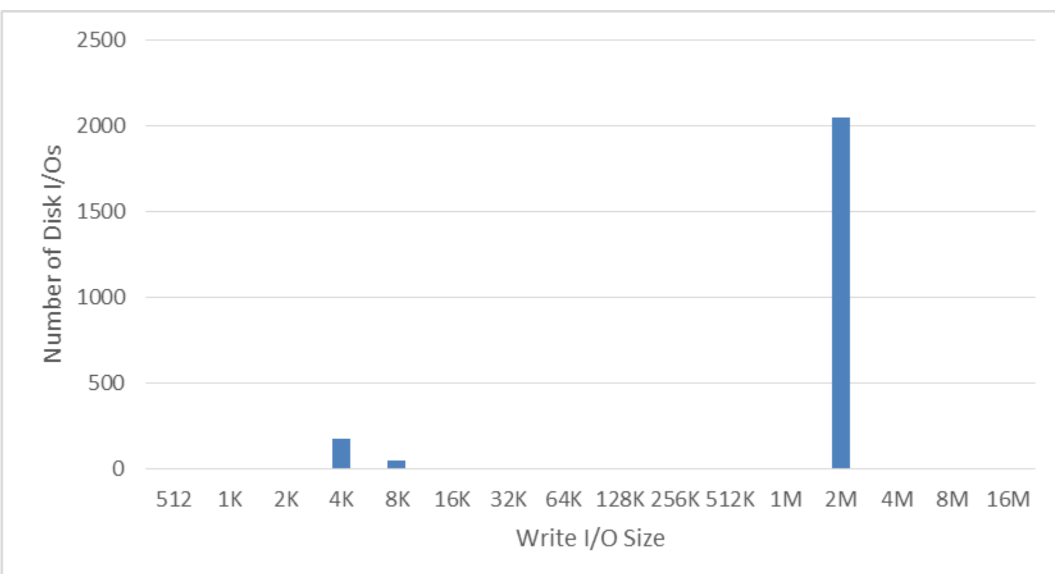
A.4.3 ZFS I/O Sizes

Using the 'zpool iostat' command, it is possible to see how the 16MB I/Os from Lustre are converted to disk I/Os:

```
$ zpool iostat -r
ssu_2ost0      sync_read     sync_write    async_read    async_write   scrub
req_size       ind    agg    ind    agg    ind    agg    ind    agg    ind    agg
----------    -----  -----  -----  -----  -----  -----  -----  -----  -----  -----
512               0      0      0      0      0      0      0      0      0      0
1K                0      0      0      0      0      0      0      0      0      0
2K                0      0      0      0      0      0      0      0      0      0
4K                0      0      0      0      0      0    171      0      0      0
8K                0      0      0      0      0      0      3     42      0      0
16K               0      0      0      0      0      0      0      0      0      0
32K               0      0      0      0      0      0      0      0      0      0
64K               0      0      0      0      0      0      0      0      0      0
128K              0      0      0      0      0      0      0      0      0      0
256K              0      0      0      0      0      0      0      0      0      0
512K              0      0      0      0      0      0      0      0      0      0
1M                0      0      0      0      0      0      0      0      0      0
2M                0      0      0      0      0      0  2.05K      0      0      0
4M                0      0      0      0      0      0      0      0      0      0
8M                0      0      0      0      0      0      0      0      0      0
16M               0      0      0      0      0      0      0      0      0      0
```

A plot of the output (Figure 5-3) clearly shows that the IOR test generated mostly 2MB write I/Os to disk.

Figure 5-3. Size Distribution of ZFS Write I/O



A.4.4 Linux Disk stats and Bandwidths

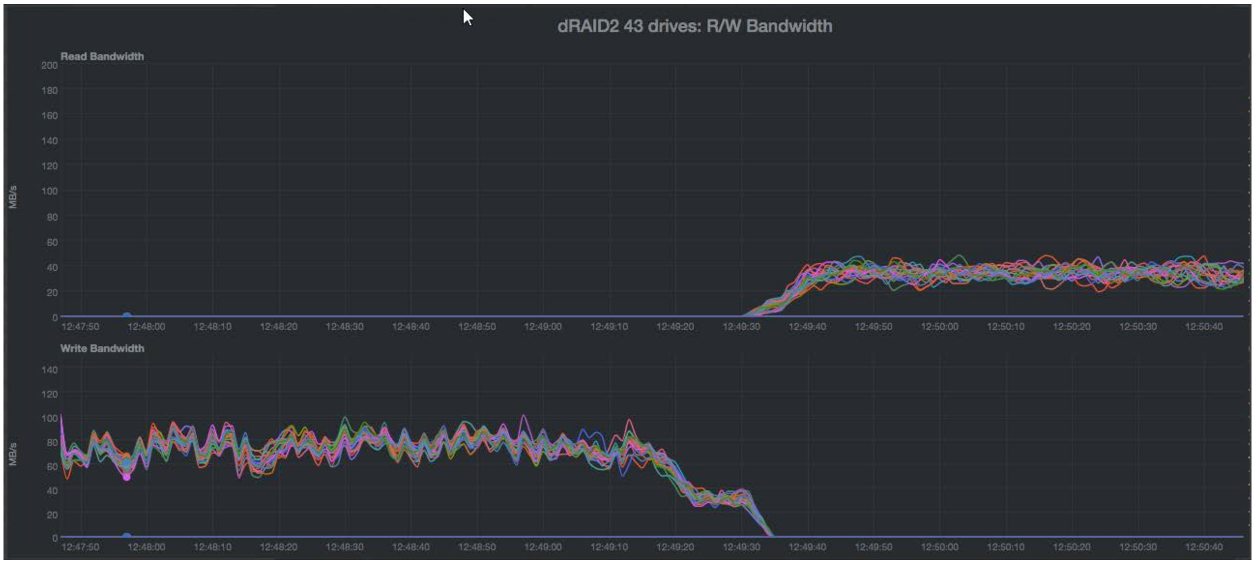
The Linux netdump utility was used to graphically display the I/O sizes and bandwidths for each disk during the IOR run. The plots show reads on top and writes on the bottom. Each line represents the data from a single drive. Time is scrolling right to left along the horizontal axis so that the oldest data are on the left of the screen. The time interval shown is at the transition from when IOR completes the writing phase of the test and begins reading the files just written.

The following figure (Figure 5-4.) shows the I/O size plot. The average write size from the 43 disks varies from 1000 KB to 1700 KB in size. Although the results above (section A.4.3) show that ZFS is sending 2MB I/Os to Linux , netdump reports the average Linux I/O size. Since ZFS writes a lot of metadata during commits, as the block pointer tree is updated at the end of each write transaction group, the average write size is expected to be less than 2MB. The read plot, however, consistently shows that Linux is reading 2000 KB from all disks.

Figure 5-4. Read/Write Disk Stats for Sequential Workload

The next figure (Figure 5-5) shows the write and read bandwidth during the same interval. The data show that all disks are writing 60-80 MB/s and reading 30-50 MB/s.

Figure 5-5. Write/Read Bandwidth for Sequential Workload



A.4.5 Linux disk stats for a random workload

The IOR test was repeated, using random file offsets for the 16MB I/Os to generate a random workload.

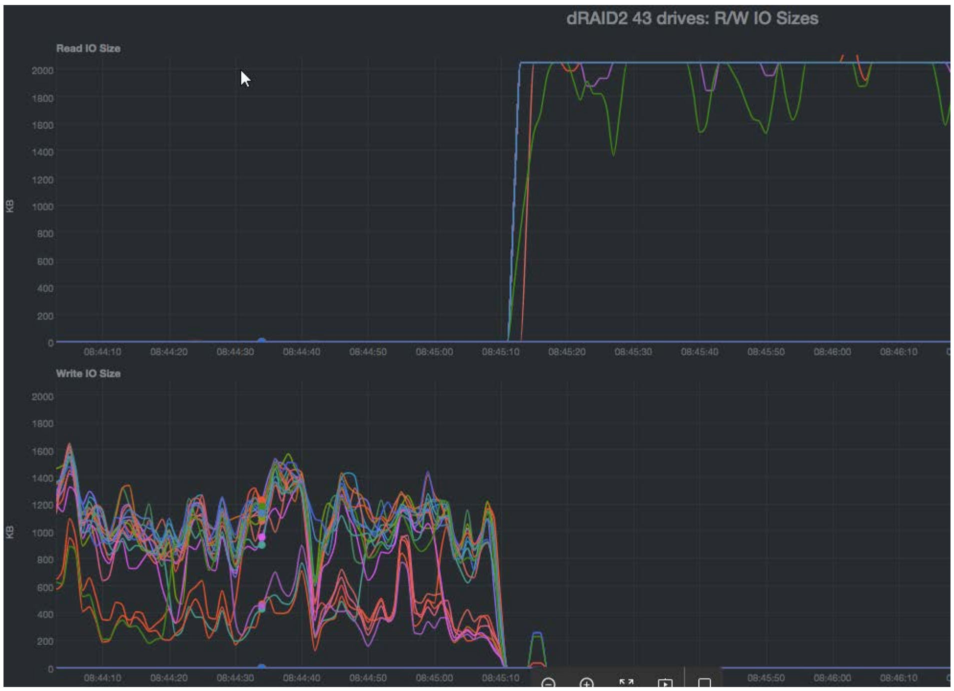
```
# mpirun -wdir /mnt/lustre -np 12 -machinefile hosts /root/natasha-bin/ior -z
-F -i 1 -s 10240 -b 16m -t 16m
IOR-3.0.1: MPI Coordinated Test of Parallel I/O

Began: Wed Jun 21 08:43:14 2017
Command line used: /root/natasha-bin/ior -z -F -i 1 -s 10240 -b 16m -t 16m
Machine: Linux node01

Test 0 started: Wed Jun 21 08:43:14 2017
Summary:
    api                 = POSIX
    test filename       = testFile
    access              = file-per-process
    ordering in a file = random offsets
    ordering inter file= no tasks offsets
    clients             = 12 (2 per node)
    repetitions         = 1
    xfersize            = 16 MiB
    blocksize           = 16 MiB
```

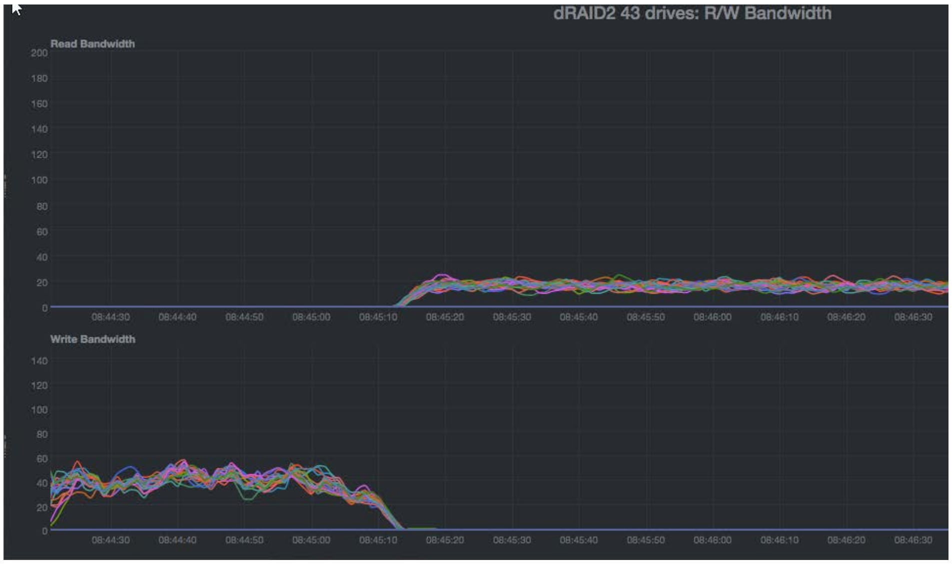
The impact of the unaligned I/O will be seen most clearly when the data is written to disk. The following diagram (Figure 5-6) shows the I/O size plots during IOR's transition from writes to reads. As above, read I/Os are consistently 2000 KB in size, as expected. Write I/Os, however, are more variable, showing a smaller size range (200-1600 KB) and greater differences between drives.

Figure 5-6. Read/Write Disk Stats for Random Workload



The bandwidth plot for the random workload (Figure 5-7) shows that, as above, the drives are writing and reading at fairly consistent rates, with no clear outliers. Nonetheless, performance per drive of 40 MB/s for writes and 20 MB/s for reads is lower than with the sequential workload above.

Figure 5-7. Write/Read Bandwidth for Random Workload



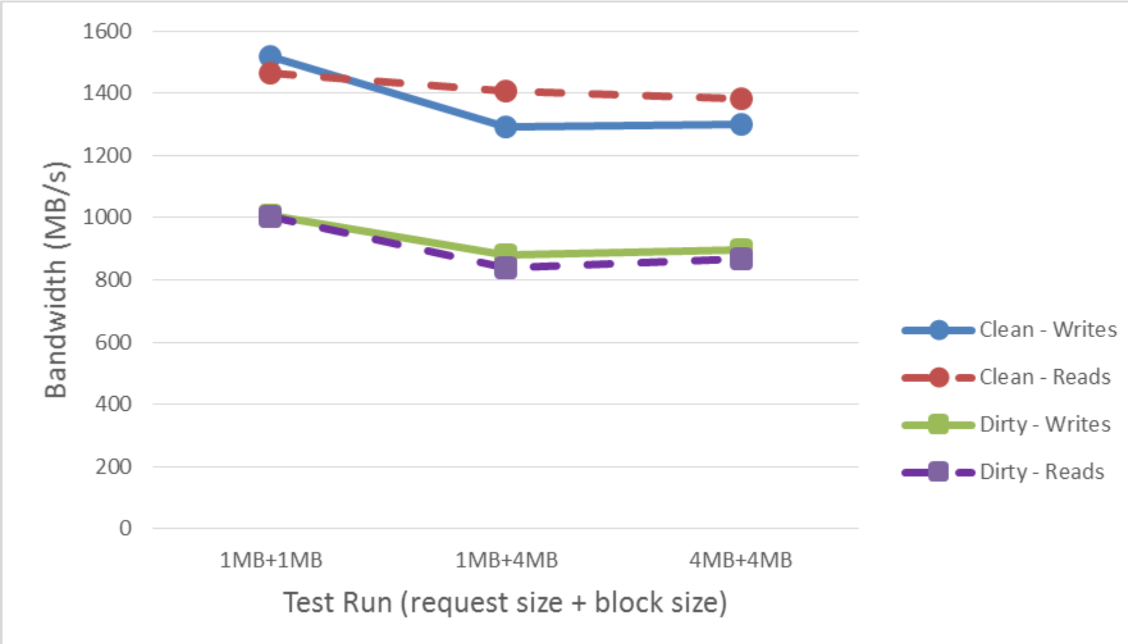
A.5 Fragmentation Improvements

Initial testing of large 16MB I/O had shown the impact of file system fragmentation, which occurred naturally as the file system aged, on performance. The metadata isolation project grew out of these early experiments. To demonstrate the benefit of metadata isolation we compared pools with and without segregation and then showed how segregation improves performance as the file system becomes fragmented.

A.5.1 File System Fragmention

When ZFS on Linux enabled support 16MB blocks, our testing found that as on-disk fragmentation increased, performance on I/O benchmarks decreased. Our early testing used a python file-ager. An iozone benchmark would be run on a clean file system, then we would run a python-based file ager to fragment the file system. The initial tool would write a concurrent combination of large files with large blocks, many smaller files 1/16th of the large block size, and random sized files. The initial tests varied the I/O request size against the ZFS block size. The results clearly show the lower performance for the dirty file systems (Figure 5-8).

Figure 5-8. Fragmentation Impact on RAIDZ1 Performance



A.5.2 Performance Improvements with Segregated Metadata

Two pools were created, one without segregation (ssu_2ost0) and one with segregation enabled (ssu_1ost1). Both pools were fragmented using the procedure described in the previous section, which left each pool with over 90% of the storage space allocated. An immediate difference could be seen from the fragmentation metric that ZFS maintains for the pool: the zpool without fragmentation had a significantly higher fragmentation score than the zpool with segregation:

- ssu_2ost0, dRAID without segregation enabled:

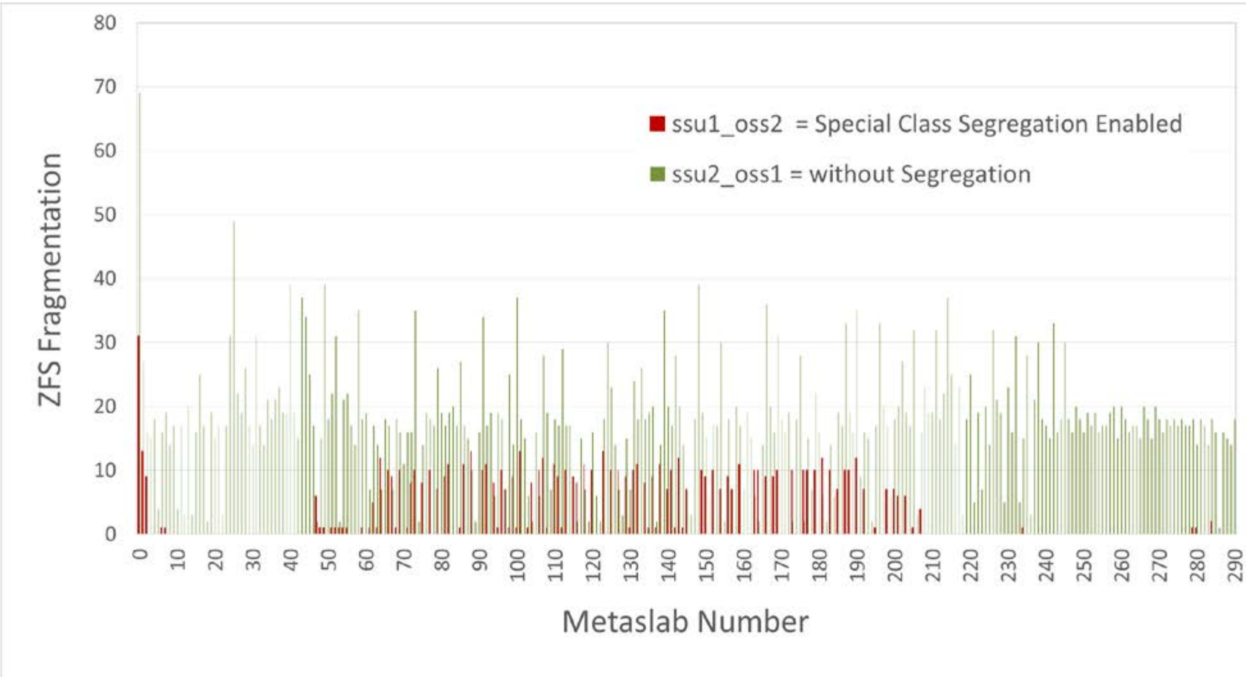
```
#zpool list
NAME        SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
ssu_2ost0  72.7T  69.6T  3.15T         -    24%    95%  1.00x  ONLINE  -
```

- ssu_1ost1, dRAID with segregation of special allocation class enabled:

```
#zpool list
NAME        SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
ssu_1ost1  72.7T  62.9T   9.9T         -     3%    92%  1.00x  ONLINE  -
```

Segregating the ZFS metadata and small block data to a separate group of metaslabs within the zpool keeps the rest of the dRAID available for efficient large block allocations. The fragmentation metric for each metaslab is available through the ‘`zdb -mm`’ command (Section A.2.3). The plot of the ZFS fragmentation metric (Figure 5-9) shows that without segregation, the metaslabs for the plain dRAID are more fragmented than the metaslabs on a dRAID with segregation enabled.

Figure 5-9. Fragmentation Comparison of Segregated and Unsegregated dRAIDs



The average ZFS fragmentation score is over 17 with many metaslabs peaking over 30. On the other hand, segregation of the metadata keeps the metslab fragmentation score below 10 (average <3). On the dRAID with the segregated VDEV, the first 20% of the metaslabs are assigned to the special class. This means that the metaslabs numbered 58 and higher (Figure 5-9) are all normal class and contain generic data larger than 32KB in size. Segregation simplifies block allocation to these metaslabs, reduces fragmentation, and improves performance.

We ran the following iozone test on the cluster using all 8 Lustre clients.

```
#iozone -t 8 -r 16m -i 0 -i 1 -s 256g
```

The test results provide aggregate throughput for all 8 nodes after writing, rewriting, reading and rereading to the Lustre servers. The results (Figure 5-10) show that without the special class enabled, performance on a fragmented file system is worse than on a file system with segregation enabled.

Figure 5-10. Performance Impact of Segregation on Fragmented File System

A.6 Examples of ZED Fault Handling using dRAID for ZFS

A.6.1 Multi-Fault Handling

This example focuses on the interaction of the Diagnosis Engine and Retire Agent with ZFS in the presence of multiple drive failures, which exercises the features described above in Section 3.

The examples show that while dRAID rebuild is in progress, the arrival of the second or third failure in the array would be saved within the Retire Agent until the current rebuild completed, upon which the Agent would retry the pending request.

To introduce the drive faults for the demonstration, we injected IO read errors using zinject to force ZED to fail the drives. The Diagnosis Engine receives these error events and once the number of faults received exceeds a failure rate threshold, the Engine will initiate a fault message. The Retire Agent then automatically begins to use the first of the dRAID spares to rebuid the failed drive.

A.6.1.1 dRAID Configuration

A triple parity dRAID was created on the cluster with a single parity group (7+3) and 3 distributed spares. We started with a clean, populated pool with all cache cleared:

```
  pool: mfault
 state: ONLINE
  scan: resilvered 368K in 0h0m1s with 0 errors on Tue Jun 27 20:11:10 2017
config:

      NAME            STATE     READ WRITE CKSUM
      mfault          ONLINE       0     0     0
        draid3-0      ONLINE       0     0     0
          sdb         ONLINE       0     0     0
          sdc         ONLINE       0     0     0
          sdd         ONLINE       0     0     0
          sde         ONLINE       0     0     0
          sdf         ONLINE       0     0     0
          sdg         ONLINE       0     0     0
          sdh         ONLINE       0     0     0
          sdi         ONLINE       0     0     0
          sdj         ONLINE       0     0     0
          sdk         ONLINE       0     0     0
          sdl         ONLINE       0     0     0
          sdm         ONLINE       0     0     0
      spares
        $draid3-0-s0  AVAIL
        $draid3-0-s1  AVAIL
        $draid3-0-s2  AVAIL

errors: No known data errors
```

A.6.1.2 First Failure

zinject() was used to send I/O read errors to drives sdb, sdg and sdm. When the Diagnosis Engine began receiving errors, the zed log showed that the Diagnosis Engine opened a failure case for each VDEV:

```
Diagnosis Engine: case opened (cc7e90a9-f96d-4937-ace2-54502acfc9ec)
Diagnosis Engine: opening case for vdev 13059866864003676862 due to
   'ereport.fs.zfs.io'
. . .
Diagnosis Engine: case opened (602adf2c-c4dc-4613-88de-de3442182e6d)
Diagnosis Engine: opening case for vdev 5296963598540981156 due to
   'ereport.fs.zfs.io'
. . .
Diagnosis Engine: case opened (ae95fc42-ccd1-4a03-924c-649ecda66de8)
Diagnosis Engine: opening case for vdev 4853321467358484743 due to
   'ereport.fs.zfs.io'
```

The Diagnosis Engine used each opened case to track the errors received for each VDEV. Eventually the first drive (sdb) accumulated enough errors that the Diagnosis Engine generated a fault event for the drive. The Retire Agent received the fault, which caused it to report the failed drive to ZFS, and then initiated the dRAID rebuild of the first distributed spare ($draid3-0-s0).

```
Diagnosis Engine: solving fault 'fault.fs.zfs.vdev.io'

zed_fault_event:
      uuid: cc7e90a9-f96d-4937-ace2-54502acfc9ec
      class: fault.fs.zfs.vdev.io
      code: ZFS-8000-FD
      certainty: 100
      scheme: zfs
      pool: 13206115887366634121
      vdev: 13059866864003676862

Diagnosis Engine: case solved (cc7e90a9-f96d-4937-ace2-54502acfc9ec)
Diagnosis Engine: removing timer (0x7f12d000d720)
Retire Agent: zfs_retire_recv: 'list.suspect'
Retire Agent: matched vdev 13059866864003676862
Retire Agent: zpool_vdev_fault: vdev 13059866864003676862 on 'mfault'
Retire Agent: zpool_vdev_replace 'sdb' with spare '$draid3-0-s0'

Diagnosis Engine: resource event 'resource.fs.zfs.statechange'
Retire Agent: zfs_retire_recv: 'resource.fs.zfs.statechange'
```

The last log messages show that ZFS had faulted the drive and sent the event through the Diagnosis Engine to the Retire Agent.

Zpool status confirms that the first drive had been faulted and that the rebuild was in progress.

```
  pool: mfault
 state: DEGRADED
status: One or more devices are faulted in response to persistent errors.
      Sufficient replicas exist for the pool to continue functioning in a
      degraded state.
action: Replace the faulted device, or use 'zpool clear' to mark the device
      repaired.
  scan: rebuild in progress since Tue Jun 27 20:03:41 2017
      2.70M scanned out of 128G at 923K/s, 40h25m to go
      1.40M rebuilt, 0.00% done
config:

      NAME                STATE     READ WRITE CKSUM
      mfault              DEGRADED     0     0     0
        draid3-0          DEGRADED     0     0     0
          spare-0         DEGRADED     0     0     0
            sdb           FAULTED     66     0     0  too many errors
            $draid3-0-s0  ONLINE       0     0     0  (repairing)
          sdc             ONLINE       0     0     0
          sdd             ONLINE       0     0     0
          sde             ONLINE       0     0     0
          sdf             ONLINE       0     0     0
          sdg             ONLINE      65     0     0  (repairing)
          sdh             ONLINE       0     0     0  (repairing)
          sdi             ONLINE       0     0     0
          sdj             ONLINE       0     0     0
          sdk             ONLINE       0     0     0
          sdl             ONLINE       0     0     0
          sdm             ONLINE      38     0     0  (repairing)
      spares
        $draid3-0-s0      INUSE     currently in use
        $draid3-0-s1      AVAIL
        $draid3-0-s2      AVAIL
errors: No known data errors
```

A.6.1.3 Second Failure

While the first rebuild was proceeding, the zinject() continued to send I/O read failures to the other two drives (sdg, sdm). Eventually, the Diagnosis Engine faulted the second drive (sdg). The Retire Agent received the fault and forwarded the fault event to ZFS. The Agent then attempted to use the second distributed spare but detected that the zpool was busy with the first rebuild and saved the spare-in request. ZFS received the fault event from the Retire Agent, then sent a state change event, which both the Diagnosis Engine and Retire Agent received.

```
Diagnosis Engine: solving fault 'fault.fs.zfs.vdev.io'

zed_fault_event:
   uuid: 602adf2c-c4dc-4613-88de-de3442182e6d
   class: fault.fs.zfs.vdev.io
   code: ZFS-8000-FD
   certainty: 100
   scheme: zfs
   pool: 1320611588736634121
   vdev: 5296963598540981156

Diagnosis Engine: case solved (602adf2c-c4dc-4613-88de-de3442182e6d)
Diagnosis Engine: removing timer (0x7f12d0045880)
Retire Agent: zfs_retire_recv: 'list.suspect'
Retire Agent: matched vdev 5296963598540981156
Retire Agent: zpool_vdev_fault: vdev 5296963598540981156 on 'mfault'
Retire Agent: zpool_vdev_replace 'sdg' with spare '$draid3-0-s1'
Retire Agent: zpool_vdev_attach 'sdg' busy. Saving request.'
Retire Agent: Saved request pool_guid 1320611588736634121 vdev_guid
   5296963598540981156.

Diagnosis Engine: resource event 'resource.fs.zfs.statechange'
Retire Agent: zfs_retire_recv: 'resource.fs.zfs.statechange'
```

After the first rebuild completed, the Retire Agent received the rebuild_finish event, then replayed the retained spare request for the second distributed spare drive ($draid3-0-s1).

```
Retire Agent: zfs_retire_recv: 'sysevent.fs.zfs.rebuild_finish'
Retire Agent: Replaying spare request pool_guid 1320611588736634121 vdev_guid
5296963598540981156.
Retire Agent: matched vdev 5296963598540981156
Retire Agent: zpool_vdev_replace 'sdg' with spare '$draid3-0-s1'
```

Initiation of this second dRAID rebuild could be seen in the zpool status.

```
  pool: mfault
```

```
 state: DEGRADED
status: One or more devices are faulted in response to persistent errors.
      Sufficient replicas exist for the pool to continue functioning in a
      degraded state.
action: Replace the faulted device, or use 'zpool clear' to mark the device
      repaired.
  scan: rebuild in progress since Tue Jun 27 20:05:43 2017
      11.6G scanned out of 128G at 1.29G/s, 0h1m to go
      762M rebuilt, 9.07% done
config:

      NAME                STATE     READ WRITE CKSUM
      mfault              DEGRADED     0     0     0
        draid3-0          DEGRADED     0     0     0
          spare-0         DEGRADED     0     0     0
            sdb           FAULTED     66     0     0  too many errors
            $draid3-0-s0  ONLINE       0     0     0  (repairing)
          sdc             ONLINE       0     0     0  (repairing)
          sdd             ONLINE       0     0     0  (repairing)
          sde             ONLINE       0     0     0  (repairing)
          sdf             ONLINE       0     0     0  (repairing)
          spare-5         DEGRADED     0     0     0
            sdg           FAULTED     65     0     0  too many errors
            $draid3-0-s1  ONLINE       0     0     0  (repairing)
          sdh             ONLINE       0     0     0  (repairing)
          sdi             ONLINE       0     0     0  (repairing)
          sdj             ONLINE       0     0     0  (repairing)
          sdk             ONLINE       0     0     0  (repairing)
          sdl             ONLINE       0     0     0  (repairing)
          sdm             FAULTED    495     0     0  too many errors
      spares
        $draid3-0-s0      INUSE     currently in use
        $draid3-0-s1      INUSE     currently in use
        $draid3-0-s2      AVAIL
errors: No known data errors
```

A.6.1.4 Third Failure

The third fault occurred while the first rebuild was in progress. The Retire Agent received the fault from the Diagnosis Engine and sent a fault event for the drive to ZFS. The Agent attempted to swap in the next available spare (still the 2nd distributed spare drive) only to find that the device was busy because a rebuild was in progress. The Retire Agent saved the request to be replayed later after the rebuild completes.

```
Diagnosis Engine: solving fault 'fault.fs.zfs.vdev.io'
```

```
zed_fault_event:
      uuid: ae95fc42-ccd1-4a03-924c-649ecda66de8
      class: fault.fs.zfs.vdev.io
      code: ZFS-8000-FD
      certainty: 100
      scheme: zfs
      pool: 13206115887366634121
      vdev: 4853321467358484743

Diagnosis Engine: case solved (ae95fc42-ccd1-4a03-924c-649ecda66de8)
Diagnosis Engine: removing timer (0x7f12d0045460)
Retire Agent: zfs_retire_recv: 'list.suspect'
Retire Agent: matched vdev 4853321467358484743
Retire Agent: zpool_vdev_fault: vdev 4853321467358484743 on 'mfault'
Retire Agent: zpool_vdev_replace 'sdm' with spare '$draid3-0-s1'
Retire Agent: zpool_vdev_attach 'sdm' busy. Saving request.'
Retire Agent: Saved request pool_guid 13206115887366634121 vdev_guid
4853321467358484743.
Diagnosis Engine: resource event 'resource.fs.zfs.statechange'
Retire Agent: zfs_retire_recv: 'resource.fs.zfs.statechange'
```

Because the fault of sdm is the second spare request saved, the drive will not be replaced until after the dRAID recovers from the fault of the second failed drive (sdg). The replacement of sdm to the third distributed spare ($draid3-0-s2) starts after the Retire Agent received the rebuild_finish for sdg.

```
Retire Agent: zfs_retire_recv: 'sysevent.fs.zfs.rebuild_finish'
Retire Agent: Replaying spare request pool_guid 13206115887366634121 vdev_guid
4853321467358484743.
Retire Agent: matched vdev 4853321467358484743
Retire Agent: zpool_vdev_replace 'sdm' with spare '$draid3-0-s2'
```

Zpool status shows that rebuild of the third failed drive was in progress.

```
  pool: mfault
 state: DEGRADED
status: One or more devices are faulted in response to persistent errors.
      Sufficient replicas exist for the pool to continue functioning in a
      degraded state.
action: Replace the faulted device, or use 'zpool clear' to mark the device
      repaired.
  scan: rebuild in progress since Tue Jun 27 20:07:44 2017
      1.96G scanned out of 128G at 1002M/s, 0h2m to go
      103M rebuilt, 1.53% done
config:
```

```
      NAME                STATE     READ WRITE CKSUM
      mfault              DEGRADED     0     0     0
        draid3-0          DEGRADED     0     0     0
          spare-0         DEGRADED     0     0     0
            sdb           FAULTED     66     0     0  too many errors
            $draid3-0-s0  ONLINE       0     0     0  (repairing)
          sdc             ONLINE       0     0     0  (repairing)
          sdd             ONLINE       0     0     0  (repairing)
          sde             ONLINE       0     0     0  (repairing)
          sdf             ONLINE       0     0     0  (repairing)
          spare-5         DEGRADED     0     0     0
            sdg           FAULTED     65     0     0  too many errors
            $draid3-0-s1  ONLINE       0     0     0  (repairing)
          sdh             ONLINE       0     0     0  (repairing)
          sdi             ONLINE       0     0     0  (repairing)
          sdj             ONLINE       0     0     0  (repairing)
          sdk             ONLINE       0     0     0  (repairing)
          sdl             ONLINE       0     0     0  (repairing)
          spare-11        DEGRADED     0     0     0
            sdm           FAULTED    495     0     0  too many errors
            $draid3-0-s2  ONLINE       0     0     0  (repairing)
      spares
        $draid3-0-s0      INUSE     currently in use
        $draid3-0-s1      INUSE     currently in use
        $draid3-0-s2      INUSE     currently in use
errors: No known data errors
```

A.6.1.5 Rebuild Complete

ZFS will send a state change event to the Diagnosis Engine after the rebuild of each drive completes to indicate that the repair is complete and the replaced drive is healthy.

```
Diagnosis Engine: resource event 'resource.fs.zfs.statechange'
Diagnosis Engine: closing case after a device statechange to healthy
Diagnosis Engine: case closed (cc7e90a9-f96d-4937-ace2-54502acfc9ec)
Diagnosis Engine: serd_destroy zfs_1253c13e38e94d09_b53df7b7fad57abe_io
Retire Agent: zfs_retire_recv: 'resource.fs.zfs.statechange'
Retire Agent: marking repaired vdev 13059866864003676862 on pool
1320611588736634121

Diagnosis Engine: resource event 'resource.fs.zfs.statechange'
Diagnosis Engine: closing case after a device statechange to healthy
Diagnosis Engine: case closed (602adf2c-c4dc-4613-88de-de3442182e6d)
Diagnosis Engine: serd_destroy zfs_1253c13e38e94d09_498298540f35c3a4_io
```

```
Retire Agent: zfs_retire_recv: 'resource.fs.zfs.statechange'
Retire Agent: marking repaired vdev 5296963598540981156 on pool
1320611588736634121

Diagnosis Engine: resource event 'resource.fs.zfs.statechange'
Diagnosis Engine: closing case after a device statechange to healthy
Diagnosis Engine: case closed (ae95fc42-ccd1-4a03-924c-649ecda66de8)
Diagnosis Engine: serd_destroy zfs_1253c13e38e94d09_435a76291aae1907_io
Retire Agent: zfs_retire_recv: 'resource.fs.zfs.statechange'
Retire Agent: marking repaired vdev 4853321467358484743 on pool
1320611588736634121
```

Upon completion of the last rebuild, zpool status showed that all three spares were in use and the pool was restored to full redundancy. Note, however, that ZFS considers use of a spare device to be a fault and the zpool status will continue to report the array to be in a degraded state until the failed drives are physically replaced, the recovered blocks are rebalanced to the replacement drives, and the distributed spare drives are restored and available for the next failure.

```
  pool: mfault
 state: DEGRADED
status: One or more devices are faulted in response to persistent errors.
      Sufficient replicas exist for the pool to continue functioning in a
      degraded state.
action: Replace the faulted device, or use 'zpool clear' to mark the device
      repaired.
  scan: rebuilt 12.8G in 0h2m25s with 0 errors on Tue Jun 27 20:10:09 2017
config:

      NAME                STATE     READ WRITE CKSUM
      mfault              DEGRADED     0     0     0
        draid3-0          DEGRADED     0     0     0
          spare-0         DEGRADED     0     0     0
            sdb           FAULTED     66     0     0  too many errors
            $draid3-0-s0  ONLINE       0     0     0
          sdc             ONLINE       0     0     0
          sdd             ONLINE       0     0     0
          sde             ONLINE       0     0     0
          sdf             ONLINE       0     0     0
          spare-5         DEGRADED     0     0     0
            sdg           FAULTED     65     0     0  too many errors
            $draid3-0-s1  ONLINE       0     0     0
          sdh             ONLINE       0     0     0
          sdi             ONLINE       0     0     0
          sdj             ONLINE       0     0     0
          sdk             ONLINE       0     0     0
```

```
          sdl             ONLINE       0     0     0
          spare-11        DEGRADED     0     0     0
            sdm           FAULTED    495     0     0  too many errors
            $draid3-0-s2  ONLINE       0     0     0
      spares
        $draid3-0-s0      INUSE     currently in use
        $draid3-0-s1      INUSE     currently in use
        $draid3-0-s2      INUSE     currently in use
errors: No known data errors
```

Appendix B. dRAID Configuration Examples

B.1 'zdb –m' for a dRAID pool without segregation

The following listing shows the status of the simple 43-drive dRAID previously described in section A.2.2. This dRAID was not configured to use metadata isolation. As a result, all metaslabs are used for all categories of ZFS storage (generic, metadata, small block).

```
[root@ssu2_oss1]# zdb -m ssu_2ost0
Metaslabs:
        vdev         0
        metaslabs  291   offset                  size              spacemap          free
        ---------------  -------------------   ---------------   ---------------   ------------
        metaslab      0   offset            0   size 4000000000   spacemap    114   free   7.36M
        metaslab      1   offset   4000006000   size 3fffffa000   spacemap    113   free   1.64G
        metaslab      2   offset   8000002000   size 3ffffe000    spacemap    112   free    861M
        metaslab      3   offset   c000008000   size 3fffff8000   spacemap    123   free   1.04G
        metaslab      4   offset  10000004000   size 3fffffc000   spacemap    122   free   1.07G
        metaslab      5   offset  14000000000   size 4000000000   spacemap    124   free    993M
        metaslab      6   offset  18000006000   size 3fffffa000   spacemap    125   free    794M
        metaslab      7   offset  1c000002000   size 3fffffe000   spacemap    126   free   1.03G
        metaslab      8   offset  20000008000   size 3fffff8000   spacemap    128   free    973M
        metaslab      9   offset  24000004000   size 3fffffc000   spacemap    127   free   1.19G
        metaslab     10   offset  28000000000   size 4000000000   spacemap    130   free   1.86G
        metaslab     11   offset  2c000006000   size 3fffffa000   spacemap    129   free   1.15G
        metaslab     12   offset  30000002000   size 3fffffe000   spacemap    132   free   1.47G
        metaslab     13   offset  34000008000   size 3fffff8000   spacemap    131   free    746M
        metaslab     14   offset  38000004000   size 3fffffc000   spacemap    134   free   1.34G
        metaslab     15   offset  3c000000000   size 4000000000   spacemap    133   free   1.25G
        metaslab     16   offset  40000006000   size 3fffffa000   spacemap    136   free   1001M
        metaslab     17   offset  44000002000   size 3fffffe000   spacemap    135   free   1.31G
        metaslab     18   offset  48000008000   size 3fffff8000   spacemap    138   free   1.03G
        metaslab     19   offset  4c000004000   size 3fffffc000   spacemap    137   free   1.01G
        metaslab     20   offset  50000000000   size 4000000000   spacemap    140   free   1.17G
        metaslab     21   offset  54000006000   size 3fffffa000   spacemap    139   free   1.13G
```

```
        metaslab     22   offset  58000002000   size 3fffffe000   spacemap    142    free    992M
        metaslab     23   offset  5c000008000   size 3fffff8000   spacemap    141    free    863M
        metaslab     24   offset  60000004000   size 3fffffc000   spacemap    144    free   7.61G
        metaslab     25   offset  64000000000   size 4000000000   spacemap    143    free   4.92G
        metaslab     26   offset  68000006000   size 3fffffa000   spacemap    145    free   29.4G
        metaslab     27   offset  6c000002000   size 3fffffe000   spacemap    147    free   17.7G
        metaslab     28   offset  70000008000   size 3fffff8000   spacemap    146    free   1.28G
        metaslab     29   offset  74000004000   size 3fffffc000   spacemap    148    free   1.05G
        metaslab     30   offset  78000000000   size 4000000000   spacemap    151    free   13.4G
        metaslab     31   offset  7c000006000   size 3fffffa000   spacemap    150    free   1.86G
        metaslab     32   offset  80000002000   size 3fffffe000   spacemap    149    free   1.42G
        metaslab     33   offset  84000008000   size 3fffff8000   spacemap    154    free   7.43G
        metaslab     34   offset  88000004000   size 3fffffc000   spacemap    153    free   21.1G
        metaslab     35   offset  8c000000000   size 4000000000   spacemap    152    free   1009M
        metaslab     36   offset  90000006000   size 3fffffa000   spacemap    156    free   15.9G
        metaslab     37   offset  94000002000   size 3fffffe000   spacemap    155    free   2.11G
        metaslab     38   offset  98000008000   size 3fffff8000   spacemap    157    free   1.59G
        metaslab     39   offset  9c000004000   size 3fffffc000   spacemap    160    free   43.9G
        metaslab     40   offset  a0000000000   size 4000000000   spacemap    159    free   3.93G
        metaslab     41   offset  a4000006000   size 3fffffa000   spacemap    158    free   1.10G
        metaslab     42   offset  a8000002000   size 3fffffe000   spacemap    163    free   27.7G
        metaslab     43   offset  ac000008000   size 3fffff8000   spacemap    162    free   2.59G
        metaslab     44   offset  b0000004000   size 3fffffc000   spacemap    161    free   2.38G
        metaslab     45   offset  b4000000000   size 4000000000   spacemap    165    free   33.6G
        metaslab     46   offset  b8000006000   size 3fffffa000   spacemap    164    free   25.1G
        metaslab     47   offset  bc000002000   size 3fffffe000   spacemap    166    free   1.25G
        metaslab     48   offset  c0000008000   size 3fffff8000   spacemap    169    free   27.1G
        metaslab     49   offset  c4000004000   size 3fffffc000   spacemap    168    free   2.46G
        metaslab     50   offset  c8000000000   size 4000000000   spacemap    167    free   40.4G
        metaslab     51   offset  cc000006000   size 3fffffa000   spacemap    171    free   49.5G
        metaslab     52   offset  d0000002000   size 3fffffe000   spacemap    170    free   2.11G
        metaslab     53   offset  d4000008000   size 3fffff8000   spacemap    172    free   1.05G
        metaslab     54   offset  d8000004000   size 3fffffc000   spacemap    174    free   28.1G
        metaslab     55   offset  dc000000000   size 4000000000   spacemap    173    free   1.34G
        metaslab     56   offset  e0000006000   size 3fffffa000   spacemap    175    free   1.44G
        metaslab     57   offset  e4000002000   size 3fffffe000   spacemap    178    free   27.5G
```

```
        metaslab     58   offset  e8000008000   size 3fffff8000   spacemap    177    free   2.59G
        metaslab     59   offset  ec000004000   size 3ffffc000    spacemap    176    free    708M
        metaslab     60   offset  f0000000000   size 4000000000   spacemap    181    free   22.6G
        metaslab     61   offset  f4000006000   size 3fffffa000   spacemap    180    free   2.70G
        metaslab     62   offset  f8000002000   size 3ffffe000    spacemap    179    free   1003M
        metaslab     63   offset  fc000008000   size 3fffff8000   spacemap    184    free   25.2G
        metaslab     64   offset 100000004000   size 3fffffc000   spacemap    183    free   2.73G
        metaslab     65   offset 104000000000   size 4000000000   spacemap    182    free    647M
        metaslab     66   offset 108000006000   size 3fffffa000   spacemap    187    free   29.2G
        metaslab     67   offset 10c000002000   size 3fffffe000   spacemap    186    free   2.49G
        metaslab     68   offset 110000008000   size 3fffff8000   spacemap    185    free   45.8G
        metaslab     69   offset 114000004000   size 3fffffc000   spacemap    189    free   26.1G
        metaslab     70   offset 118000000000   size 4000000000   spacemap    188    free   1.97G
        metaslab     71   offset 11c000006000   size 3fffffa000   spacemap    190    free   1.07G
        metaslab     72   offset 120000002000   size 3fffffe000   spacemap    193    free   27.5G
        metaslab     73   offset 124000008000   size 3fffff8000   spacemap    192    free   2.74G
        metaslab     74   offset 128000004000   size 3fffffc000   spacemap    191    free   1.77G
        metaslab     75   offset 12c000000000   size 4000000000   spacemap    196    free   28.1G
        metaslab     76   offset 130000006000   size 3fffffa000   spacemap    195    free   40.2G
        metaslab     77   offset 134000002000   size 3fffffe000   spacemap    194    free    777M
        metaslab     78   offset 138000008000   size 3fffff8000   spacemap    198    free   26.7G
        metaslab     79   offset 13c000004000   size 3fffffc000   spacemap    197    free   1.78G
        metaslab     80   offset 140000000000   size 4000000000   spacemap    199    free   35.1G
        metaslab     81   offset 144000006000   size 3fffffa000   spacemap    201    free   27.9G
        metaslab     82   offset 148000002000   size 3fffffe000   spacemap    200    free   41.0G
        metaslab     83   offset 14c000008000   size 3fffff8000   spacemap    202    free   38.2G
        metaslab     84   offset 150000004000   size 3fffffc000   spacemap    203    free   17.9G
        metaslab     85   offset 154000000000   size 4000000000   spacemap    204    free   1.70G
        metaslab     86   offset 158000006000   size 3fffffa000   spacemap    205    free   1.84G
        metaslab     87   offset 15c000002000   size 3fffffe000   spacemap    208    free   27.3G
        metaslab     88   offset 160000008000   size 3fffff8000   spacemap    207    free   3.12G
        metaslab     89   offset 164000004000   size 3fffffc000   spacemap    206    free   1.51G
        metaslab     90   offset 168000000000   size 4000000000   spacemap    211    free   25.8G
        metaslab     91   offset 16c000006000   size 3fffffa000   spacemap    210    free   3.29G
        metaslab     92   offset 170000002000   size 3fffffe000   spacemap    209    free   2.14G
        metaslab     93   offset 174000008000   size 3fffff8000   spacemap    214    free   22.3G
```

```
        metaslab     94   offset 178000004000   size 3fffffc000   spacemap    213    free    2.64G
        metaslab     95   offset 17c000000000   size 4000000000   spacemap    212    free    1.52G
        metaslab     96   offset 180000006000   size 3fffffa000   spacemap    217    free    22.4G
        metaslab     97   offset 184000002000   size 3ffffffe000  spacemap    216    free    2.64G
        metaslab     98   offset 188000008000   size 3fffff8000   spacemap    215    free    1.79G
        metaslab     99   offset 18c000004000   size 3fffffc000   spacemap    220    free    26.5G
        metaslab    100   offset 190000000000   size 4000000000   spacemap    219    free    2.58G
        metaslab    101   offset 194000006000   size 3fffffa000   spacemap    218    free    1.72G
        metaslab    102   offset 198000002000   size 3ffffffe000  spacemap    223    free    27.0G
        metaslab    103   offset 19c000008000   size 3fffff8000   spacemap    222    free    2.66G
        metaslab    104   offset 1a0000004000   size 3fffffc000   spacemap    221    free    1.24G
        metaslab    105   offset 1a4000000000   size 4000000000   spacemap    226    free    26.1G
        metaslab    106   offset 1a8000006000   size 3fffffa000   spacemap    225    free    2.84G
        metaslab    107   offset 1ac000002000   size 3ffffffe000  spacemap    224    free    1.55G
        metaslab    108   offset 1b0000008000   size 3fffff8000   spacemap    229    free    44.5G
        metaslab    109   offset 1b4000004000   size 3fffffc000   spacemap    228    free    2.96G
        metaslab    110   offset 1b8000000000   size 4000000000   spacemap    227    free    1.53G
        metaslab    111   offset 1bc000006000   size 3fffffa000   spacemap    231    free    28.0G
        metaslab    112   offset 1c0000002000   size 3ffffffe000  spacemap    230    free    2.73G
        metaslab    113   offset 1c4000008000   size 3fffff8000   spacemap    232    free    1.62G
        metaslab    114   offset 1c8000004000   size 3fffffc000   spacemap    235    free    46.8G
        metaslab    115   offset 1cc000000000   size 4000000000   spacemap    234    free    4.07G
        metaslab    116   offset 1d0000006000   size 3fffffa000   spacemap    233    free    1.59G
        metaslab    117   offset 1d4000002000   size 3ffffffe000  spacemap    238    free    25.9G
        metaslab    118   offset 1d8000008000   size 3fffff8000   spacemap    237    free    2.92G
        metaslab    119   offset 1dc000004000   size 3fffffc000   spacemap    236    free    1.54G
        metaslab    120   offset 1e0000000000   size 4000000000   spacemap    241    free    28.7G
        metaslab    121   offset 1e4000006000   size 3fffffa000   spacemap    240    free    2.55G
        metaslab    122   offset 1e8000002000   size 3ffffffe000  spacemap    239    free    1.38G
        metaslab    123   offset 1ec000008000   size 3fffff8000   spacemap    244    free    28.7G
        metaslab    124   offset 1f0000004000   size 3fffffc000   spacemap    243    free    4.02G
        metaslab    125   offset 1f4000000000   size 4000000000   spacemap    242    free    1.64G
        metaslab    126   offset 1f8000006000   size 3fffffa000   spacemap    247    free    28.9G
        metaslab    127   offset 1fc000002000   size 3ffffffe000  spacemap    246    free    3.00G
        metaslab    128   offset 200000008000   size 3fffff8000   spacemap    245    free    2.16G
        metaslab    129   offset 204000004000   size 3fffffc000   spacemap    250    free    26.7G
```

```
        metaslab   130   offset 208000000000   size 4000000000   spacemap   249   free   2.63G
        metaslab   131   offset 20c000006000   size 3fffffa000   spacemap   248   free   38.5G
        metaslab   132   offset 210000002000   size 3ffffffe000  spacemap   252   free   17.1G
        metaslab   133   offset 214000008000   size 3fffff8000   spacemap   251   free   2.03G
        metaslab   134   offset 218000004000   size 3fffffc000   spacemap   253   free   2.19G
        metaslab   135   offset 21c000000000   size 4000000000   spacemap   255   free   43.2G
        metaslab   136   offset 220000006000   size 3fffffa000   spacemap   254   free   2.22G
        metaslab   137   offset 224000002000   size 3ffffffe000  spacemap   256   free   1.58G
        metaslab   138   offset 228000008000   size 3fffff8000   spacemap   259   free   25.6G
        metaslab   139   offset 22c000004000   size 3fffffc000   spacemap   258   free   2.58G
        metaslab   140   offset 230000000000   size 4000000000   spacemap   257   free   35.2G
        metaslab   141   offset 234000006000   size 3fffffa000   spacemap   261   free   26.5G
        metaslab   142   offset 238000002000   size 3ffffffe000  spacemap   260   free   1.74G
        metaslab   143   offset 23c000008000   size 3fffff8000   spacemap   262   free   1.92G
        metaslab   144   offset 240000004000   size 3fffffc000   spacemap   265   free   25.5G
        metaslab   145   offset 244000000000   size 4000000000   spacemap   264   free   3.25G
        metaslab   146   offset 248000006000   size 3fffffa000   spacemap   263   free   2.17G
        metaslab   147   offset 24c000002000   size 3ffffffe000  spacemap   268   free   44.3G
        metaslab   148   offset 250000008000   size 3fffff8000   spacemap   267   free   3.64G
        metaslab   149   offset 254000004000   size 3fffffc000   spacemap   266   free   1.56G
        metaslab   150   offset 258000000000   size 4000000000   spacemap   271   free   27.5G
        metaslab   151   offset 25c000006000   size 3fffffa000   spacemap   270   free   2.11G
        metaslab   152   offset 260000002000   size 3ffffffe000  spacemap   269   free   1.91G
        metaslab   153   offset 264000008000   size 3fffff8000   spacemap   274   free   26.9G
        metaslab   154   offset 268000004000   size 3fffffc000   spacemap   273   free   3.66G
        metaslab   155   offset 26c000000000   size 4000000000   spacemap   272   free   1.43G
        metaslab   156   offset 270000006000   size 3fffffa000   spacemap   277   free   25.6G
        metaslab   157   offset 274000002000   size 3ffffffe000  spacemap   276   free   2.62G
        metaslab   158   offset 278000008000   size 3fffff8000   spacemap   275   free   1.89G
        metaslab   159   offset 27c000004000   size 3fffffc000   spacemap   280   free   27.4G
        metaslab   160   offset 280000000000   size 4000000000   spacemap   279   free   3.38G
        metaslab   161   offset 284000006000   size 3fffffa000   spacemap   278   free   1.39G
        metaslab   162   offset 288000002000   size 3ffffffe000  spacemap   283   free   26.4G
        metaslab   163   offset 28c000008000   size 3fffff8000   spacemap   282   free   2.76G
        metaslab   164   offset 290000004000   size 3fffffc000   spacemap   281   free   1.72G
        metaslab   165   offset 294000000000   size 4000000000   spacemap   286   free   23.1G
```

```
        metaslab    166   offset 298000006000   size 3fffffa000   spacemap    285   free   3.01G
        metaslab    167   offset 29c000002000   size 3fffffe000   spacemap    284   free   48.2G
        metaslab    168   offset 2a0000008000   size 3ffffff8000   spacemap    288   free   26.0G
        metaslab    169   offset 2a4000004000   size 3fffffc000   spacemap    287   free   2.25G
        metaslab    170   offset 2a8000000000   size 4000000000   spacemap    289   free   38.5G
        metaslab    171   offset 2ac000006000   size 3fffffa000   spacemap    291   free   27.0G
        metaslab    172   offset 2b0000002000   size 3fffffe000   spacemap    290   free   40.5G
        metaslab    173   offset 2b4000008000   size 3ffffff8000   spacemap    292   free    993M
        metaslab    174   offset 2b8000004000   size 3fffffc000   spacemap    294   free   18.7G
        metaslab    175   offset 2bc000000000   size 4000000000   spacemap    293   free   1.82G
        metaslab    176   offset 2c0000006000   size 3fffffa000   spacemap    295   free   1.64G
        metaslab    177   offset 2c4000002000   size 3fffffe000   spacemap    298   free   3.25G
        metaslab    178   offset 2c8000008000   size 3ffffff8000   spacemap    297   free   2.57G
        metaslab    179   offset 2cc000004000   size 3fffffc000   spacemap    296   free   1.36G
        metaslab    180   offset 2d0000000000   size 4000000000   spacemap    301   free   6.21G
        metaslab    181   offset 2d4000006000   size 3fffffa000   spacemap    300   free   2.58G
        metaslab    182   offset 2d8000002000   size 3fffffe000   spacemap    299   free   1.73G
        metaslab    183   offset 2dc000008000   size 3ffffff8000   spacemap    304   free   27.6G
        metaslab    184   offset 2e0000004000   size 3fffffc000   spacemap    303   free   2.64G
        metaslab    185   offset 2e4000000000   size 4000000000   spacemap    302   free   1.11G
        metaslab    186   offset 2e8000006000   size 3fffffa000   spacemap    307   free   6.57G
        metaslab    187   offset 2ec000002000   size 3fffffe000   spacemap    306   free   2.35G
        metaslab    188   offset 2f0000008000   size 3ffffff8000   spacemap    305   free   1.52G
        metaslab    189   offset 2f4000004000   size 3fffffc000   spacemap    310   free   6.39G
        metaslab    190   offset 2f8000000000   size 4000000000   spacemap    309   free   2.17G
        metaslab    191   offset 2fc000006000   size 3fffffa000   spacemap    308   free   4.71G
        metaslab    192   offset 300000002000   size 3fffffe000   spacemap    313   free   4.68G
        metaslab    193   offset 304000008000   size 3ffffff8000   spacemap    312   free   4.54G
        metaslab    194   offset 308000004000   size 3fffffc000   spacemap    311   free   1.39G
        metaslab    195   offset 30c000000000   size 4000000000   spacemap    316   free   3.55G
        metaslab    196   offset 310000006000   size 3fffffa000   spacemap    315   free   2.23G
        metaslab    197   offset 314000002000   size 3fffffe000   spacemap    314   free   1.38G
        metaslab    198   offset 318000008000   size 3ffffff8000   spacemap    319   free   25.4G
        metaslab    199   offset 31c000004000   size 3fffffc000   spacemap    318   free   3.14G
        metaslab    200   offset 320000000000   size 4000000000   spacemap    317   free   39.3G
        metaslab    201   offset 324000006000   size 3fffffa000   spacemap    321   free   26.6G
```

```
        metaslab    202   offset 328000002000   size 3fffffe000   spacemap    320    free    1.99G
        metaslab    203   offset 32c000008000   size 3fffff8000   spacemap    322    free    1.68G
        metaslab    204   offset 330000004000   size 3fffffc000   spacemap    325    free    28.8G
        metaslab    205   offset 334000000000   size 4000000000   spacemap    324    free    2.95G
        metaslab    206   offset 338000006000   size 3fffffa000   spacemap    323    free    1.78G
        metaslab    207   offset 33c000002000   size 3fffffe000   spacemap    328    free    24.4G
        metaslab    208   offset 340000008000   size 3fffff8000   spacemap    327    free    29.8G
        metaslab    209   offset 344000004000   size 3fffffc000   spacemap    326    free    8.55G
        metaslab    210   offset 348000000000   size 4000000000   spacemap    330    free    26.9G
        metaslab    211   offset 34c000006000   size 3fffffa000   spacemap    329    free    1.99G
        metaslab    212   offset 350000002000   size 3fffffe000   spacemap    331    free    1.83G
        metaslab    213   offset 354000008000   size 3fffff8000   spacemap    334    free    26.7G
        metaslab    214   offset 358000004000   size 3fffffc000   spacemap    333    free    2.35G
        metaslab    215   offset 35c000000000   size 4000000000   spacemap    332    free    2.82G
        metaslab    216   offset 360000006000   size 3fffffa000   spacemap    337    free    26.5G
        metaslab    217   offset 364000002000   size 3fffffe000   spacemap    336    free    8.60G
        metaslab    218   offset 368000008000   size 3fffff8000   spacemap    335    free    2.16G
        metaslab    219   offset 36c000004000   size 3fffffc000   spacemap    340    free    21.6G
        metaslab    220   offset 370000000000   size 4000000000   spacemap    339    free    7.50G
        metaslab    221   offset 374000006000   size 3fffffa000   spacemap    338    free    1.52G
        metaslab    222   offset 378000002000   size 3fffffe000   spacemap    343    free    42.7G
        metaslab    223   offset 37c000008000   size 3fffff8000   spacemap    342    free    3.92G
        metaslab    224   offset 380000004000   size 3fffffc000   spacemap    341    free    2.40G
        metaslab    225   offset 384000000000   size 4000000000   spacemap    345    free    26.4G
        metaslab    226   offset 388000006000   size 3fffffa000   spacemap    344    free    2.50G
        metaslab    227   offset 38c000002000   size 3fffffe000   spacemap    346    free    32.9G
        metaslab    228   offset 390000008000   size 3fffff8000   spacemap    348    free    44.8G
        metaslab    229   offset 394000004000   size 3fffffc000   spacemap    347    free    1.67G
        metaslab    230   offset 398000000000   size 4000000000   spacemap    349    free    2.25G
        metaslab    231   offset 39c000006000   size 3fffffa000   spacemap    352    free    27.8G
        metaslab    232   offset 3a0000002000   size 3fffffe000   spacemap    351    free    3.14G
        metaslab    233   offset 3a4000008000   size 3fffff8000   spacemap    350    free    2.73G
        metaslab    234   offset 3a8000004000   size 3fffffc000   spacemap    355    free    24.6G
        metaslab    235   offset 3ac000000000   size 4000000000   spacemap    354    free    4.52G
        metaslab    236   offset 3b0000006000   size 3fffffa000   spacemap    353    free    1.96G
        metaslab    237   offset 3b4000002000   size 3fffffe000   spacemap    358    free    34.8G
```

```
        metaslab    238   offset 3b8000008000   size 3fffff8000   spacemap    357   free   2.18G
        metaslab    239   offset 3bc000004000   size 3fffffc000   spacemap    356   free   26.0G
        metaslab    240   offset 3c0000000000   size 4000000000   spacemap    359   free   1.15G
        metaslab    241   offset 3c4000006000   size 3fffffa000   spacemap    362   free   20.4G
        metaslab    242   offset 3c8000002000   size 3ffffffe000  spacemap    361   free   2.05G
        metaslab    243   offset 3cc000008000   size 3fffff8000   spacemap    360   free    642M
        metaslab    244   offset 3d0000004000   size 3fffffc000   spacemap    365   free   15.7G
        metaslab    245   offset 3d4000000000   size 4000000000   spacemap    364   free   1.97G
        metaslab    246   offset 3d8000006000   size 3fffffa000   spacemap    363   free   1.09G
        metaslab    247   offset 3dc000002000   size 3ffffffe000  spacemap    368   free   2.12G
        metaslab    248   offset 3e0000008000   size 3fffff8000   spacemap    367   free   1.25G
        metaslab    249   offset 3e4000004000   size 3fffffc000   spacemap    366   free   1.27G
        metaslab    250   offset 3e8000000000   size 4000000000   spacemap    371   free    737M
        metaslab    251   offset 3ec000006000   size 3fffffa000   spacemap    370   free   1.31G
        metaslab    252   offset 3f0000002000   size 3ffffffe000  spacemap    369   free   1.43G
        metaslab    253   offset 3f4000008000   size 3fffff8000   spacemap    374   free   1.10G
        metaslab    254   offset 3f8000004000   size 3fffffc000   spacemap    373   free   1.12G
        metaslab    255   offset 3fc000000000   size 4000000000   spacemap    372   free   1.06G
        metaslab    256   offset 400000006000   size 3fffffa000   spacemap    377   free    829M
        metaslab    257   offset 404000002000   size 3ffffffe000  spacemap    376   free    820M
        metaslab    258   offset 408000008000   size 3fffff8000   spacemap    375   free    774M
        metaslab    259   offset 40c000004000   size 3fffffc000   spacemap    380   free   1014M
        metaslab    260   offset 410000000000   size 4000000000   spacemap    379   free    648M
        metaslab    261   offset 414000006000   size 3fffffa000   spacemap    378   free   1.15G
        metaslab    262   offset 418000002000   size 3ffffffe000  spacemap    383   free   1.04G
        metaslab    263   offset 41c000008000   size 3fffff8000   spacemap    382   free   1002M
        metaslab    264   offset 420000004000   size 3fffffc000   spacemap    381   free    955M
        metaslab    265   offset 424000000000   size 4000000000   spacemap    386   free    896M
        metaslab    266   offset 428000006000   size 3fffffa000   spacemap    385   free    986M
        metaslab    267   offset 42c000002000   size 3ffffffe000  spacemap    384   free   1.15G
        metaslab    268   offset 430000008000   size 3fffff8000   spacemap    389   free    706M
        metaslab    269   offset 434000004000   size 3fffffc000   spacemap    388   free   1.10G
        metaslab    270   offset 438000000000   size 4000000000   spacemap    387   free    831M
        metaslab    271   offset 43c000006000   size 3fffffa000   spacemap    392   free    903M
        metaslab    272   offset 440000002000   size 3ffffffe000  spacemap    391   free    777M
        metaslab    273   offset 444000008000   size 3fffff8000   spacemap    390   free   1.04G
```

```
        metaslab    274   offset 448000004000   size 3fffffc000   spacemap    393   free    794M
        metaslab    275   offset 44c000000000   size 4000000000   spacemap    394   free    695M
        metaslab    276   offset 450000006000   size 3fffffa000   spacemap    395   free   1.01G
        metaslab    277   offset 454000002000   size 3ffffffe000  spacemap    398   free    899M
        metaslab    278   offset 458000008000   size 3ffffff8000  spacemap    397   free    927M
        metaslab    279   offset 45c000004000   size 3fffffc000   spacemap    396   free    823M
        metaslab    280   offset 460000000000   size 4000000000   spacemap    401   free    821M
        metaslab    281   offset 464000006000   size 3fffffa000   spacemap    400   free    505M
        metaslab    282   offset 468000002000   size 3ffffffe000  spacemap    399   free    926M
        metaslab    283   offset 46c000008000   size 3ffffff8000  spacemap    404   free    960M
        metaslab    284   offset 470000004000   size 3fffffc000   spacemap    403   free    867M
        metaslab    285   offset 474000000000   size 4000000000   spacemap    402   free    509M
        metaslab    286   offset 478000006000   size 3fffffa000   spacemap    407   free    928M
        metaslab    287   offset 47c000002000   size 3ffffffe000  spacemap    406   free    658M
        metaslab    288   offset 480000008000   size 3ffffff8000  spacemap    405   free    931M
        metaslab    289   offset 484000004000   size 3fffffc000   spacemap    409   free    726M
        metaslab    290   offset 488000000000   size 4000000000   spacemap    408   free    583M
```

B.2 'zdb –m' for a dRAID pool with segregation enabled

The following listing shows the status of the 43-drive hybrid dRAID described in section A.2.2. This dRAID was configured to use metadata isolation with segregation enabled. The listing includes an extra column of that describes the class assignment for each metaslab. The first 20% are reserved for the special class and will contain both metadata and small block categories. The normal class will be used first for data larger than 32KB in size. When the special class metaslabs are consumed, small block I/O will spill over into the normal class.

```
[root@ssu1_oss2]# zdb -m ssu_lost1
Metaslabs:
        vdev          0   segregate
        metaslabs   291   offset                 size                spacemap          free           class
        ---------------   -------------------    ---------------     ---------------   ------------   --------
        metaslab      0   offset             0   size 4000000000     spacemap    115   free    122G    special
        metaslab      1   offset    4000000000   size 4000000000     spacemap    114   free    208G    special
```

```
        metaslab      2   offset   8000001000   size 3ffffff000   spacemap    113   free     221G   special
        metaslab      3   offset   c000001000   size 3ffffff000   spacemap      4   free     256G   special
        metaslab      4   offset  10000002000   size 3fffffe000   spacemap      3   free     256G   special
        metaslab      5   offset  14000000000   size 4000000000   spacemap      2   free     256G   special
        metaslab      6   offset  18000000000   size 4000000000   spacemap      7   free     256G   special
        metaslab      7   offset  1c000001000   size 3ffffff000   spacemap      6   free     256G   special
        metaslab      8   offset  20000001000   size 3ffffff000   spacemap      5   free     256G   special
        metaslab      9   offset  24000002000   size 3fffffe000   spacemap      0   free     256G    ----
        metaslab     10   offset  28000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     11   offset  2c000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     12   offset  30000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     13   offset  34000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     14   offset  38000002000   size 3fffffe000   spacemap      0   free     256G    ----
        metaslab     15   offset  3c000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     16   offset  40000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     17   offset  44000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     18   offset  48000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     19   offset  4c000002000   size 3fffffe000   spacemap      0   free     256G    ----
        metaslab     20   offset  50000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     21   offset  54000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     22   offset  58000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     23   offset  5c000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     24   offset  60000002000   size 3fffffe000   spacemap      0   free     256G    ----
        metaslab     25   offset  64000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     26   offset  68000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     27   offset  6c000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     28   offset  70000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     29   offset  74000002000   size 3fffffe000   spacemap      0   free     256G    ----
        metaslab     30   offset  78000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     31   offset  7c000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     32   offset  80000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     33   offset  84000001000   size 3ffffff000   spacemap      0   free     256G    ----
        metaslab     34   offset  88000002000   size 3fffffe000   spacemap      0   free     256G    ----
        metaslab     35   offset  8c000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     36   offset  90000000000   size 4000000000   spacemap      0   free     256G    ----
        metaslab     37   offset  94000001000   size 3ffffff000   spacemap      0   free     256G    ----
```

```
metaslab     38   offset  98000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     39   offset  9c000002000   size 3fffffe000   spacemap     0   free    256G    ----
metaslab     40   offset  a0000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     41   offset  a4000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     42   offset  a8000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     43   offset  ac000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     44   offset  b0000002000   size 3fffffe000   spacemap     0   free    256G    ----
metaslab     45   offset  b4000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     46   offset  b8000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     47   offset  bc000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     48   offset  c0000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     49   offset  c4000002000   size 3fffffe000   spacemap     0   free    256G    ----
metaslab     50   offset  c8000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     51   offset  cc000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     52   offset  d0000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     53   offset  d4000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     54   offset  d8000002000   size 3fffffe000   spacemap     0   free    256G    ----
metaslab     55   offset  dc000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     56   offset  e0000000000   size 4000000000   spacemap     0   free    256G    ----
metaslab     57   offset  e4000001000   size 3ffffff000   spacemap     0   free    256G    ----
metaslab     58   offset  e8000008000   size 3ffff8000   spacemap   123   free   7.70G    normal
metaslab     59   offset  ec000004000   size 3ffffc000   spacemap   125   free   1.43G    normal
metaslab     60   offset  f0000000000   size 4000000000   spacemap   124   free   1.58G    normal
metaslab     61   offset  f4000006000   size 3ffffa000   spacemap   126   free   1.27G    normal
metaslab     62   offset  f8000002000   size 3fffffe000   spacemap   127   free   1.66G    normal
metaslab     63   offset  fc000008000   size 3ffff8000   spacemap   128   free   2.05G    normal
metaslab     64   offset 100000004000   size 3ffffc000   spacemap   129   free   2.23G    normal
metaslab     65   offset 104000000000   size 4000000000   spacemap   130   free   2.01G    normal
metaslab     66   offset 108000006000   size 3ffffa000   spacemap   131   free   1.59G    normal
metaslab     67   offset 10c000002000   size 3fffffe000   spacemap   132   free   1.08G    normal
metaslab     68   offset 110000008000   size 3ffff8000   spacemap   133   free   1.29G    normal
metaslab     69   offset 114000004000   size 3ffffc000   spacemap   134   free   1.59G    normal
metaslab     70   offset 118000000000   size 4000000000   spacemap   135   free   1.52G    normal
metaslab     71   offset 11c000006000   size 3ffffa000   spacemap   136   free    949M    normal
metaslab     72   offset 120000002000   size 3fffffe000   spacemap   137   free   1.59G    normal
metaslab     73   offset 124000008000   size 3ffff8000   spacemap   138   free   14.5G    normal
```

```
metaslab     74   offset 128000004000   size 3fffffc000   spacemap    139   free   1.85G    normal
metaslab     75   offset 12c000000000   size 4000000000   spacemap    141   free   24.1G    normal
metaslab     76   offset 130000006000   size 3fffffa000   spacemap    140   free   1.07G    normal
metaslab     77   offset 134000002000   size 3ffffffe000  spacemap    142   free   19.9G    normal
metaslab     78   offset 138000008000   size 3ffffff8000  spacemap    144   free   14.9G    normal
metaslab     79   offset 13c000004000   size 3fffffc000   spacemap    143   free   1.49G    normal
metaslab     80   offset 140000000000   size 4000000000   spacemap    145   free   22.2G    normal
metaslab     81   offset 144000006000   size 3fffffa000   spacemap    146   free   1.11G    normal
metaslab     82   offset 148000002000   size 3ffffffe000  spacemap    147   free   1.34G    normal
metaslab     83   offset 14c000008000   size 3ffffff8000  spacemap    148   free   36.2G    normal
metaslab     84   offset 150000004000   size 3fffffc000   spacemap    150   free   23.0G    normal
metaslab     85   offset 154000000000   size 4000000000   spacemap    149   free    946M    normal
metaslab     86   offset 158000006000   size 3fffffa000   spacemap    151   free   38.8G    normal
metaslab     87   offset 15c000002000   size 3ffffffe000  spacemap    152   free   1.04G    normal
metaslab     88   offset 160000008000   size 3ffffff8000  spacemap    153   free   25.3G    normal
metaslab     89   offset 164000004000   size 3fffffc000   spacemap    154   free   1.66G    normal
metaslab     90   offset 168000000000   size 4000000000   spacemap    155   free   34.4G    normal
metaslab     91   offset 16c000006000   size 3fffffa000   spacemap    156   free   1.51G    normal
metaslab     92   offset 170000002000   size 3ffffffe000  spacemap    157   free   27.5G    normal
metaslab     93   offset 174000008000   size 3ffffff8000  spacemap    159   free   23.8G    normal
metaslab     94   offset 178000004000   size 3fffffc000   spacemap    158   free   1.08G    normal
metaslab     95   offset 17c000000000   size 4000000000   spacemap    160   free   1.54G    normal
metaslab     96   offset 180000006000   size 3fffffa000   spacemap    161   free   1.46G    normal
metaslab     97   offset 184000002000   size 3ffffffe000  spacemap    162   free   24.6G    normal
metaslab     98   offset 188000008000   size 3ffffff8000  spacemap    163   free   1.07G    normal
metaslab     99   offset 18c000004000   size 3fffffc000   spacemap    165   free   43.3G    normal
metaslab    100   offset 190000000000   size 4000000000   spacemap    164   free    633M    normal
metaslab    101   offset 194000006000   size 3fffffa000   spacemap    166   free   1.27G    normal
metaslab    102   offset 198000002000   size 3ffffffe000  spacemap    167   free   20.8G    normal
metaslab    103   offset 19c000008000   size 3ffffff8000  spacemap    169   free   24.3G    normal
metaslab    104   offset 1a0000004000   size 3fffffc000   spacemap    168   free   11.4G    normal
metaslab    105   offset 1a4000000000   size 4000000000   spacemap    170   free   35.3G    normal
metaslab    106   offset 1a8000006000   size 3fffffa000   spacemap    171   free   1.45G    normal
metaslab    107   offset 1ac000002000   size 3ffffffe000  spacemap    172   free   24.3G    normal
metaslab    108   offset 1b0000008000   size 3ffffff8000  spacemap    174   free   36.1G    normal
metaslab    109   offset 1b4000004000   size 3fffffc000   spacemap    173   free   1.54G    normal
```

```
        metaslab    110   offset 1b8000000000   size 4000000000   spacemap    175   free   28.0G    normal
        metaslab    111   offset 1bc000006000   size 3fffffa000   spacemap    176   free   1.43G    normal
        metaslab    112   offset 1c0000002000   size 3ffffffe000  spacemap    177   free   35.6G    normal
        metaslab    113   offset 1c4000008000   size 3fffff8000   spacemap    178   free   1.09G    normal
        metaslab    114   offset 1c8000004000   size 3fffffc000   spacemap    179   free   1.43G    normal
        metaslab    115   offset 1cc000000000   size 4000000000   spacemap    181   free   28.4G    normal
        metaslab    116   offset 1d0000006000   size 3fffffa000   spacemap    180   free   1.21G    normal
        metaslab    117   offset 1d4000002000   size 3fffffe000   spacemap    182   free   22.7G    normal
        metaslab    118   offset 1d8000008000   size 3fffff8000   spacemap    184   free   45.0G    normal
        metaslab    119   offset 1dc000004000   size 3fffffc000   spacemap    183   free   1.63G    normal
        metaslab    120   offset 1e0000000000   size 4000000000   spacemap    185   free   19.0G    normal
        metaslab    121   offset 1e4000006000   size 3fffffa000   spacemap    186   free   22.6G    normal
        metaslab    122   offset 1e8000002000   size 3fffffe000   spacemap    188   free   29.7G    normal
        metaslab    123   offset 1ec000008000   size 3fffff8000   spacemap    187   free   1.62G    normal
        metaslab    124   offset 1f0000004000   size 3fffffc000   spacemap    189   free   26.4G    normal
        metaslab    125   offset 1f4000000000   size 4000000000   spacemap    190   free   1.45G    normal
        metaslab    126   offset 1f8000006000   size 3fffffa000   spacemap    191   free   25.4G    normal
        metaslab    127   offset 1fc000002000   size 3fffffe000   spacemap    193   free   32.9G    normal
        metaslab    128   offset 200000008000   size 3fffff8000   spacemap    192   free   1.02G    normal
        metaslab    129   offset 204000004000   size 3fffffc000   spacemap    194   free   21.2G    normal
        metaslab    130   offset 208000000000   size 4000000000   spacemap    195   free   17.8G    normal
        metaslab    131   offset 20c000006000   size 3fffffa000   spacemap    196   free   22.8G    normal
        metaslab    132   offset 210000002000   size 3fffffe000   spacemap    197   free   1.47G    normal
        metaslab    133   offset 214000008000   size 3fffff8000   spacemap    198   free   1.22G    normal
        metaslab    134   offset 218000004000   size 3fffffc000   spacemap    199   free   31.4G    normal
        metaslab    135   offset 21c000000000   size 4000000000   spacemap    200   free   1.20G    normal
        metaslab    136   offset 220000006000   size 3fffffa000   spacemap    201   free   24.0G    normal
        metaslab    137   offset 224000002000   size 3fffffe000   spacemap    202   free   1.38G    normal
        metaslab    138   offset 228000008000   size 3fffff8000   spacemap    203   free   22.8G    normal
        metaslab    139   offset 22c000004000   size 3fffffc000   spacemap    204   free   1.40G    normal
        metaslab    140   offset 230000000000   size 4000000000   spacemap    206   free   29.0G    normal
        metaslab    141   offset 234000006000   size 3fffffa000   spacemap    205   free   1.60G    normal
        metaslab    142   offset 238000002000   size 3fffffe000   spacemap    207   free   22.2G    normal
        metaslab    143   offset 23c000008000   size 3fffff8000   spacemap    209   free   20.7G    normal
        metaslab    144   offset 240000004000   size 3fffffc000   spacemap    208   free   1.22G    normal
        metaslab    145   offset 244000000000   size 4000000000   spacemap    210   free   30.2G    normal
```

```
        metaslab    146   offset 248000006000   size 3fffffa000   spacemap    211   free    1.32G    normal
        metaslab    147   offset 24c000002000   size 3ffffe000    spacemap    212   free   20.4G    normal
        metaslab    148   offset 250000008000   size 3fffff8000   spacemap    213   free    1.32G    normal
        metaslab    149   offset 254000004000   size 3fffffc000   spacemap    214   free   15.6G    normal
        metaslab    150   offset 258000000000   size 4000000000   spacemap    215   free    1.01G    normal
        metaslab    151   offset 25c000006000   size 3ffffa000    spacemap    216   free   34.1G    normal
        metaslab    152   offset 260000002000   size 3ffffe000    spacemap    218   free   18.7G    normal
        metaslab    153   offset 264000008000   size 3fffff8000   spacemap    217   free    1.21G    normal
        metaslab    154   offset 268000004000   size 3fffffc000   spacemap    219   free   36.5G    normal
        metaslab    155   offset 26c000000000   size 4000000000   spacemap    220   free    1.44G    normal
        metaslab    156   offset 270000006000   size 3ffffa000    spacemap    221   free   33.3G    normal
        metaslab    157   offset 274000002000   size 3ffffe000    spacemap    222   free    1.17G    normal
        metaslab    158   offset 278000008000   size 3fffff8000   spacemap    223   free    1.58G    normal
        metaslab    159   offset 27c000004000   size 3fffffc000   spacemap    224   free    1.33G    normal
        metaslab    160   offset 280000000000   size 4000000000   spacemap    225   free   21.4G    normal
        metaslab    161   offset 284000006000   size 3ffffa000    spacemap    227   free   24.5G    normal
        metaslab    162   offset 288000002000   size 3ffffe000    spacemap    226   free    1.50G    normal
        metaslab    163   offset 28c000008000   size 3fffff8000   spacemap    228   free   19.1G    normal
        metaslab    164   offset 290000004000   size 3fffffc000   spacemap    229   free    1.33G    normal
        metaslab    165   offset 294000000000   size 4000000000   spacemap    230   free   32.4G    normal
        metaslab    166   offset 298000006000   size 3ffffa000    spacemap    231   free     907M    normal
        metaslab    167   offset 29c000002000   size 3ffffe000    spacemap    232   free   17.1G    normal
        metaslab    168   offset 2a0000008000   size 3fffff8000   spacemap    234   free   29.6G    normal
        metaslab    169   offset 2a4000004000   size 3fffffc000   spacemap    233   free    1.46G    normal
        metaslab    170   offset 2a8000000000   size 4000000000   spacemap    235   free   16.2G    normal
        metaslab    171   offset 2ac000006000   size 3ffffa000    spacemap    236   free    1.36G    normal
        metaslab    172   offset 2b0000002000   size 3ffffe000    spacemap    237   free    1.07G    normal
        metaslab    173   offset 2b4000008000   size 3fffff8000   spacemap    238   free     895M    normal
        metaslab    174   offset 2b8000004000   size 3fffffc000   spacemap    239   free   15.6G    normal
        metaslab    175   offset 2bc000000000   size 4000000000   spacemap    241   free   17.4G    normal
        metaslab    176   offset 2c0000006000   size 3ffffa000    spacemap    240   free    1.03G    normal
        metaslab    177   offset 2c4000002000   size 3ffffe000    spacemap    242   free   17.7G    normal
        metaslab    178   offset 2c8000008000   size 3fffff8000   spacemap    243   free    1.29G    normal
        metaslab    179   offset 2cc000004000   size 3fffffc000   spacemap    244   free   22.8G    normal
        metaslab    180   offset 2d0000000000   size 4000000000   spacemap    246   free   18.4G    normal
        metaslab    181   offset 2d4000006000   size 3ffffa000    spacemap    245   free     950M    normal
```

```
        metaslab    182   offset 2d8000002000   size 3fffffe000   spacemap    247   free    932M    normal
        metaslab    183   offset 2dc000008000   size 3ffff8000   spacemap    248   free    987M    normal
        metaslab    184   offset 2e0000004000   size 3fffffc000   spacemap    249   free   14.2G    normal
        metaslab    185   offset 2e4000000000   size 4000000000   spacemap    250   free   1.09G    normal
        metaslab    186   offset 2e8000006000   size 3fffffa000   spacemap    251   free    582M    normal
        metaslab    187   offset 2ec000002000   size 3fffffe000   spacemap    252   free   16.0G    normal
        metaslab    188   offset 2f0000008000   size 3ffff8000   spacemap    254   free   13.0G    normal
        metaslab    189   offset 2f4000004000   size 3fffffc000   spacemap    253   free   1.40G    normal
        metaslab    190   offset 2f8000000000   size 4000000000   spacemap    255   free   15.6G    normal
        metaslab    191   offset 2fc000006000   size 3fffffa000   spacemap    256   free   10.9G    normal
        metaslab    192   offset 300000002000   size 3fffffe000   spacemap    257   free   32.9G    normal
        metaslab    193   offset 304000008000   size 3ffff8000   spacemap    258   free   1.12G    normal
        metaslab    194   offset 308000004000   size 3fffffc000   spacemap    259   free   16.6G    normal
        metaslab    195   offset 30c000000000   size 4000000000   spacemap    260   free   1.34G    normal
        metaslab    196   offset 310000006000   size 3fffffa000   spacemap    261   free   28.0G    normal
        metaslab    197   offset 314000002000   size 3fffffe000   spacemap    262   free    892M    normal
        metaslab    198   offset 318000008000   size 3ffff8000   spacemap    263   free   17.8G    normal
        metaslab    199   offset 31c000004000   size 3fffffc000   spacemap    265   free   16.7G    normal
        metaslab    200   offset 320000000000   size 4000000000   spacemap    264   free   1.25G    normal
        metaslab    201   offset 324000006000   size 3fffffa000   spacemap    266   free   30.3G    normal
        metaslab    202   offset 328000002000   size 3fffffe000   spacemap    267   free   1.26G    normal
        metaslab    203   offset 32c000008000   size 3ffff8000   spacemap    268   free   29.4G    normal
        metaslab    204   offset 330000004000   size 3fffffc000   spacemap    269   free   1.26G    normal
        metaslab    205   offset 334000000000   size 4000000000   spacemap    270   free    757M    normal
        metaslab    206   offset 338000006000   size 3fffffa000   spacemap    271   free   1.59G    normal
        metaslab    207   offset 33c000002000   size 3fffffe000   spacemap    272   free    571M    normal
        metaslab    208   offset 340000008000   size 3ffff8000   spacemap    273   free   1.37G    normal
        metaslab    209   offset 344000004000   size 3fffffc000   spacemap    274   free   27.6G    normal
        metaslab    210   offset 348000000000   size 4000000000   spacemap    275   free    717M    normal
        metaslab    211   offset 34c000006000   size 3fffffa000   spacemap    276   free   24.7G    normal
        metaslab    212   offset 350000002000   size 3fffffe000   spacemap    278   free   33.1G    normal
        metaslab    213   offset 354000008000   size 3ffff8000   spacemap    277   free   1.32G    normal
        metaslab    214   offset 358000004000   size 3fffffc000   spacemap    279   free   25.6G    normal
        metaslab    215   offset 35c000000000   size 4000000000   spacemap    280   free   1.29G    normal
        metaslab    216   offset 360000006000   size 3fffffa000   spacemap    281   free   1.38G    normal
        metaslab    217   offset 364000002000   size 3fffffe000   spacemap    282   free   1.23G    normal
```

```
metaslab   218   offset 368000008000   size 3fffff8000   spacemap   283   free   22.7G   normal
metaslab   219   offset 36c000004000   size 3ffffc000   spacemap   284   free   1.33G   normal
metaslab   220   offset 370000000000   size 4000000000   spacemap   285   free    670M   normal
metaslab   221   offset 374000006000   size 3ffffa000   spacemap   287   free   37.0G   normal
metaslab   222   offset 378000002000   size 3ffffe000   spacemap   286   free   22.8G   normal
metaslab   223   offset 37c000008000   size 3fffff8000   spacemap   288   free   15.6G   normal
metaslab   224   offset 380000004000   size 3ffffc000   spacemap   289   free   22.5G   normal
metaslab   225   offset 384000000000   size 4000000000   spacemap   290   free   22.9G   normal
metaslab   226   offset 388000006000   size 3ffffa000   spacemap   291   free   15.2G   normal
metaslab   227   offset 38c000002000   size 3ffffe000   spacemap   292   free   22.3G   normal
metaslab   228   offset 390000008000   size 3fffff8000   spacemap   293   free   1.07G   normal
metaslab   229   offset 394000004000   size 3ffffc000   spacemap   294   free    948M   normal
metaslab   230   offset 398000000000   size 4000000000   spacemap   295   free    901M   normal
metaslab   231   offset 39c000006000   size 3ffffa000   spacemap   296   free    815M   normal
metaslab   232   offset 3a0000002000   size 3ffffe000   spacemap   297   free   1.33G   normal
metaslab   233   offset 3a4000008000   size 3fffff8000   spacemap   298   free   1.15G   normal
metaslab   234   offset 3a8000004000   size 3ffffc000   spacemap   299   free   1.08G   normal
metaslab   235   offset 3ac000000000   size 4000000000   spacemap   300   free    442M   normal
metaslab   236   offset 3b0000006000   size 3ffffa000   spacemap   301   free    712M   normal
metaslab   237   offset 3b4000002000   size 3ffffe000   spacemap   302   free    994M   normal
metaslab   238   offset 3b8000008000   size 3fffff8000   spacemap   303   free   1.17G   normal
metaslab   239   offset 3bc000004000   size 3ffffc000   spacemap   304   free   1.14G   normal
metaslab   240   offset 3c0000000000   size 4000000000   spacemap   305   free    285M   normal
metaslab   241   offset 3c4000006000   size 3ffffa000   spacemap   306   free    947M   normal
metaslab   242   offset 3c8000002000   size 3ffffe000   spacemap   307   free   1.34G   normal
metaslab   243   offset 3cc000008000   size 3fffff8000   spacemap   308   free    490M   normal
metaslab   244   offset 3d0000004000   size 3ffffc000   spacemap   309   free   23.0G   normal
metaslab   245   offset 3d4000000000   size 4000000000   spacemap   310   free   1.47G   normal
metaslab   246   offset 3d8000006000   size 3ffffa000   spacemap   311   free   21.5G   normal
metaslab   247   offset 3dc000002000   size 3ffffe000   spacemap   312   free    726M   normal
metaslab   248   offset 3e0000008000   size 3fffff8000   spacemap   313   free   1.24G   normal
metaslab   249   offset 3e4000004000   size 3ffffc000   spacemap   314   free   20.9G   normal
metaslab   250   offset 3e8000000000   size 4000000000   spacemap   316   free    946M   normal
metaslab   251   offset 3ec000006000   size 3ffffa000   spacemap   315   free   26.0G   normal
metaslab   252   offset 3f0000002000   size 3ffffe000   spacemap   317   free    677M   normal
metaslab   253   offset 3f4000008000   size 3fffff8000   spacemap   318   free   1.37G   normal
```

```
        metaslab    254   offset 3f8000004000   size 3fffffc000   spacemap   319   free   1.28G    normal
        metaslab    255   offset 3fc000000000   size 4000000000   spacemap   320   free   14.0G    normal
        metaslab    256   offset 400000006000   size 3fffffa000   spacemap   322   free    874M    normal
        metaslab    257   offset 404000002000   size 3ffffffe000  spacemap   321   free   22.3G    normal
        metaslab    258   offset 408000008000   size 3ffffff8000  spacemap   323   free   1.06G    normal
        metaslab    259   offset 40c000004000   size 3fffffc000   spacemap   324   free   1.00G    normal
        metaslab    260   offset 410000000000   size 4000000000   spacemap   326   free   23.0G    normal
        metaslab    261   offset 414000006000   size 3fffffa000   spacemap   325   free    670M    normal
        metaslab    262   offset 418000002000   size 3ffffffe000  spacemap   327   free    506M    normal
        metaslab    263   offset 41c000008000   size 3ffffff8000  spacemap   328   free   1.25G    normal
        metaslab    264   offset 420000004000   size 3fffffc000   spacemap   329   free    948M    normal
        metaslab    265   offset 424000000000   size 4000000000   spacemap   330   free   17.1G    normal
        metaslab    266   offset 428000006000   size 3fffffa000   spacemap   331   free    887M    normal
        metaslab    267   offset 42c000002000   size 3ffffffe000  spacemap   332   free    846M    normal
        metaslab    268   offset 430000008000   size 3ffffff8000  spacemap   333   free   1.06G    normal
        metaslab    269   offset 434000004000   size 3fffffc000   spacemap   334   free   1.28G    normal
        metaslab    270   offset 438000000000   size 4000000000   spacemap   335   free    793M    normal
        metaslab    271   offset 43c000006000   size 3fffffa000   spacemap   336   free   23.5G    normal
        metaslab    272   offset 440000002000   size 3ffffffe000  spacemap   337   free   20.3G    normal
        metaslab    273   offset 444000008000   size 3ffffff8000  spacemap   338   free   1.06G    normal
        metaslab    274   offset 448000004000   size 3fffffc000   spacemap   339   free    840M    normal
        metaslab    275   offset 44c000000000   size 4000000000   spacemap   340   free   20.4G    normal
        metaslab    276   offset 450000006000   size 3fffffa000   spacemap   341   free   1000M    normal
        metaslab    277   offset 454000002000   size 3ffffffe000  spacemap   342   free   1.04G    normal
        metaslab    278   offset 458000008000   size 3ffffff8000  spacemap   343   free   17.7G    normal
        metaslab    279   offset 45c000004000   size 3fffffc000   spacemap   344   free   1.23G    normal
        metaslab    280   offset 460000000000   size 4000000000   spacemap   345   free    905M    normal
        metaslab    281   offset 464000006000   size 3fffffa000   spacemap   346   free   22.1G    normal
        metaslab    282   offset 468000002000   size 3ffffffe000  spacemap   347   free   19.7G    normal
        metaslab    283   offset 46c000008000   size 3ffffff8000  spacemap   348   free   1.15G    normal
        metaslab    284   offset 470000004000   size 3fffffc000   spacemap   349   free   12.7G    normal
        metaslab    285   offset 474000000000   size 4000000000   spacemap   350   free   21.9G    normal
        metaslab    286   offset 478000006000   size 3fffffa000   spacemap   351   free   19.0G    normal
        metaslab    287   offset 47c000002000   size 3ffffffe000  spacemap   352   free   1.16G    normal
        metaslab    288   offset 480000008000   size 3ffffff8000  spacemap   353   free    912M    normal
        metaslab    289   offset 484000004000   size 3fffffc000   spacemap   354   free    902M    normal
```

```
        metaslab    290   offset 488000000000   size 4000000000   spacemap    355   free   1.35G    normal
```

B.3 draidcfg output for the 80 drive demonstration (80.nvl)

The following is the complete listing of the base permuation table created for the dRAID configuration shown in the demonstration of arbitrary pool configuration (section A.1.1). Each line represents the random ordering for the permutation of the 80 drives in the array.

```
# draidcfg -r 80.nvl
dRAID3 vdev of 80 child drives: 7 x (8 data + 3 parity) and 3 distributed spare
Using 64 base permutations
 23,54,38,76,61,14,34,48, 9,31,52,10, 3,41,46,70, 1, 6,59,47,28,32,29,49,30,22,27,11,44,20,56, 5,74,
   8,50,15,62,66,33,67,16,65,36,71,75,18,68,21,69,26,64,60,55,42,43,63,35,37,24, 7,17,45, 0, 2,58,78,57,13,12,72,73,
   4,19,25,51,79,39,53,77,40,
 41,54,75,48, 2,57,36, 8,76,44, 5, 3,22,30,61,69,47,28,13, 0, 6,71,34,55,33,46,70,79,66,45,27,74,18,25,60,72,11,50,68,
   1,53,32,19,64,40,51, 4,31,17,62,42,39,26,56, 7,16,24,12,38,15,78,35,37,67, 9,23,20,49,10,43,14,59,77,29,63,73,58,52,21,65,
 14,65,43, 9,16,53,46,69,17,40,20, 3,47,70,28,39,54, 5,12,24,78, 2,49,61,11,51,75,79,41,50,73,34,18,21,25,52,44,22,32,77, 8,59,15,
   7,74,66, 0,71,45,56, 4,36,58,23,68, 6,67,42,29,64,26,33,72,10,37,13, 1,76,60,38,48,31,63,27,35,62,55,19,30,57,
 2,56,48,51,68,15,75,41,58,35,50,14,36,16,63,77,30,69,11,10,26, 7,62,19,24,44,28,37,31,43,64,25,49,32,54,53, 9,76,39,57,33,74,
   8,34,27,23, 3,40,72,59,67,55,65,47,66, 1,71,61,46,18,17,29,79,38,12,70,22,45,78,60, 5, 6,21,73,13, 0,42,20, 4,52,
 64,76,20, 7,34,21,63,13, 0,47,51,41,59,57,74, 6,25,71,54,33,35,46,19,15,43, 8,23,18,24,61,10,39,72,27,26, 9,62,17,53,78,
   2,58,29,60,77,44,36,66,70,22,67,75,65,69,30, 4,40,14,42,45,38,49,32,11,31,16,28,79, 3,68,56,73, 1,48, 5,52,12,55,50,37,
 62,33,67,58,38,57,61,24, 3,47, 0,37,53,72,40,39,35,10,20,60,43,41,69,55,23,21,59,25,13,28, 9,12,51,19,52,27,63,45, 2,31,46,15, 5,
   7,14,68, 1,76,78,50,29,26, 6,42,22,56,11,64,16, 4,49,79,73,74,54,36, 8,77,65,17,75,66,44,71,70,32,34,18,48,30,
 10,57,30,46,42,55,34,16,52,49,44,36,53,18,79,21,38,77,60,39,45, 8,19,24,68,17,73,63,66,70,65, 7, 4,37,61, 6,64,12,
   9,26,28,14,78,31,41,27,11,33,51, 3,29,35,74,32,23,58,76,13, 0,22,15,69,47, 1,56, 5,72,67,54,50,43,48,59,25, 2,75,62,40,71,20,
 59,53,27,26,72,23,33,56,66,10,73,52,51, 8,24,18,11,68, 6,77,45,19,37, 2, 4,76,47,17,34,62,49, 0,50,28,74,22,21,15,78,
   7,25,20,40,32,35,38,31,71,57,65,12,16,13,48,43, 1,54,58,36, 9,63,64,79, 5,42,44,75,14,39,55,60,29,30,46,61,70,41,69,67, 3,
 40,14,16,31, 5,63,69,53,43,37,73,50,77,20,29,61,41,48,45,67,15,55,47,79,60,25,76,54,12,57,46,56,35, 1, 7,65,22,11,34,26,13, 70,
   27,72, 2,74,19, 8,28,23,66,62,71,33,64,18,17, 3, 0,49, 6,36,75,59,78,30,32,58,52,24, 4,21,10,68,39,51, 9,38,44,42,
 53,55,59,77,64,39,40,62,76,16,74, 5,26,23,66,47,21, 0, 6,60,69,27,11,58,72,34,73,45,38, 3,43,49,15, 9,19, 1,79,35,67,31,44,75,
   46,68,50,71, 2, 8,48,65,54,14,78,63,41,13,10,29,12,32,20,57,30,25,24,51,36,18,56,33, 4,70,37,61,17,28,52,42, 7,22,
 47,54,51, 3,49,45,32,71,68,26,31,65,30, 8,74, 9,76,78,46,25,38,53,60,19,11,50, 6,33,58,37,39, 7,20, 2, 0, 5,75,28, 63,48,44,
   40,34,41,17,15,67,16,66,13,22,72,73,21,35,36,77,12,70, 1, 4,10,69,23,52,43,79,56,59,27,62,57,55, 14,64,24,18,42,29,61,
 33,65,36,19,47,41,53, 9,26,17,66, 1, 5,77,69,59, 6,27, 7,14,71,68,12,25,28,10,13,18,61,51,22,54,34,31,48,57,32,67,49,62,
   8,23,40,58,39,79,50,74,45,44,73,64,30, 0, 2,56,75,21,29,43,63,76,72,60,46, 4,42,20,70,16, 3,11,52,55,38,78,37,24,15,35,
 75,79,64,78,22,61,28, 2,50,51,21, 5,46,72,16,15,70, 3,49,32,36,26,60,27, 0,24,52,56, 7,43,12,73,42,30,45,17,48,10,33,74,37,35,44,
   71,29,68,53,23,13,59,38,34,19,18, 8,62,65,66,39,76,20,55,77,58,25,40,41,11,31,47,57, 9,14, 4,67, 1,54,69,63, 6,
```

```
12,76,64,36,37,28,44,57,49,13,39,73,58, 5,27,52,33,15,26,56,51,66,14,63,23,71,43,62,65,11,35,74,16,18, 0, 7, 1,
   6,34,60,47,46,29,20,79, 3,10,53,42,78,68,31,21,50,54,70, 4,61,55,38,30,59,69, 8,32,77, 9,22,41,24,48,75,19,17,25,67, 2,40,45,72,
19, 2,64, 8,37,58,25,70,33,23,28,68,52,69,34,30,27,50, 7,56,62,47,51,57,45,61,65,67,31,75,79,43,42,13,29,53,66,72,77,35,26,
   5,48,15,38,10, 1,59,14,16,11,40, 3,20,76,63,49,74,41,39, 0,55,46,54,78,12, 4,22,71, 9,21,36,44,60,18,73,17,32,24, 6,
73,55,35,71,46,40,26,32,29,23,13,15,30, 0,72,68,22,64,25, 1,56,76,43,36,44, 8,27,39,18,63,41, 9,19,11,58,28,50,24, 2,77,
   7,53,34,66,49,65,78,67,59,69,57,48,70,79,33, 5,61,60,51,62, 3,14,75,16,17, 4,52,74,10,20,45,12,47,37,54,31, 6,38,21,42,
0,43,38,78,21,73,18,16, 3,79,70,24,47,74,12,67,63,30,41, 6,34,27,76,72,25,65, 9,37,55,39,31, 4,71, 8,10,58,44,11,35,36,23,46,
   48,53,52,69,32,20,59,66,26,14,62,61,57,50,15,13,17,29,60,56,68,33, 5,75,54,51,28,42,45, 2,40,22,49, 7,19, 1,77,64,
74,19,37,28,34,69,59,31,78,13,61,65,54,40,75,73,70,38,22, 6,63,10,27,48, 4,46,14,21,41, 3,15,20,76,26,77,62,60,
   8,66,45,23,55,29,50,67,11,52,25, 5, 2,36, 0,30,51,33,49,43,44,53,39,32, 1,79,71,72,12,47,58, 9,24,42,16,57,35,56, 7,68,17,18,64,
65,10, 5,51,26, 4,68,22, 7,23,55, 1,21,34,61,75,20,76, 9,74,62,48,54,73,35,13,58, 8,44,53,33,38,42,60,64,17,36,15,32,29,67,66,
   41,52,63,31,47,79, 3, 0,24,49,78,72,18,12,14,46,37,11,77,39,56,30,59,69, 2,25, 6,16,71,28,50,45,57,19,27,40,43,70,
14,79,16,51, 9,50,41,15, 2, 8,32,75,21,43,11,26,65,36,27,47,38,17,67, 3,71,45,60,42, 1,54,22, 4,20, 6,40,76,74,56,12,61,28,
   0,25,78,55,23,18,44, 5,73,52,29,77,57,34,24,58,19, 7,46,37,69,35,62,66,13,53,59,70,64,39,63,48,33,10,31,68,49,72,30,
30,52, 4,58,14,78,62, 5,76,29,36,28,63,64,74,56, 3,32,39,33,18,48,65,10,68,50,66, 0,16,57,26, 9,60,37,23,17,34,25,67,69,
   8,79,77,53,15,73,44,71,12,59, 1,51, 2,11,35, 6,47,22,46,75,38,55,49, 7,61,54,19,41,13,40,27,31,42,70,72,20,21,43,45,24,
20,21,44, 0,30,28,46, 6, 7,40,13,76,72,37,53, 8,61,57,18,35,12,78,31,17,29,79,70, 2,26,77,50,25,41,23,47, 1,34,16,69,68,10,
   3,74,59,14,55,54,60,27,49, 9,39,73,65,42,67,15,33,56,58,11,64,22,24,43, 4,36,66,38,62,51,48, 5,32,63,52,45,75,71,19,
11,31,79,26,14, 9,27,62, 1,39,29,54, 6, 5,41,28,22,65, 7,34,57,77,59,73,42,32,46,25,38,63,74,47,17,60,72,67,33,64,53,40,
   66,12,48,15,71,56,23, 3,76,44,30, 4,16,49,37,24,55,52,58,61,51,70,35,43,18,69,13,75, 2, 8,36,45,19,50, 0,10,68,78,20,21,
66,31,41,34,44,77,79,75,33,18,22,16,27,19,40,17,57, 8, 0,26,50,28,15,37,29,49,24, 6, 9,13,53,60,73,71,25,52,78,10,58,65,11, 3,62,
   1, 4,38,70,43, 5,12,20,61,63,47,23,55, 7,74,35,76,48,68,67,59,30,42,72,46,39,54,69,51,36,56,64,45,32,21,14, 2,
15,41, 7,18,34,77,36, 0,45,66,22,59,56,42,48, 8,31,61,43,30,70,13,52,60,49,75,46,53,14,64,28,16, 3,58, 2,35,26,67,72,44,79,
   29,51,74, 1, 5,55, 4,17,71,73,65,27,54,38,78,23,76,68,47,32,19,40,69,21,24,39,33,37,62,10,11, 9,20,63,25,50,12,57, 6,
74,76,51, 8,19,78,32,46, 9,63,67,49,75,26,16, 6,66,25,30,53,56,37,77,22, 1,17,45,65,59,12,68,31,55,27,23,34,44,14,11,48,54,18,36,
   64, 0,70, 3,21,39, 4,62,79,20,40,57,15,35,60,38,61,10,69,41,43,24,50,73,47,33, 7,71,13,72,42,58,28, 2,52, 5,29,
4,39,24,66,41,72,29,25, 2,55, 6,43,44,52,45,75,19, 7,22,11,20,16,63,49,59,60,26,35,54,70,64, 3,56,42,58,71,48,57,51,77,18,76,61,
   9,62,36,53,38,79, 1,50,74,47,15,14,34, 8,78,40,10,73,13,12,30,28, 5,33,17,27,37, 0,65,67,23,46,31,21,69,32,68,
27,46,61,37,41, 9,74, 7,79,73,67, 3,25,45,33,10,59,49,52,77,78,43,53, 4, 0,17,35,21,14,60,44,54,32,24,63,42,72,39,62,36,
   40,64,66,34,75, 6,30,69,38, 5,51,47,28,22,18,57,31,71,15,23,68,20,12, 1,56,76, 8,26,55, 2,70,29,13,65,48,58,11,16,50,19,
18,13, 9,72,41,35,20,11,10,70,43,37,67,73,56,33,52,46,29,30,45,61, 1, 4, 7,16,55,23,40,66,36,48,71,63,34,28,64,12,79,
   8,14,75,50,26,60,74,44,69,59,15, 3, 2,21,65,27,22,76,77,53,51,32,31,19,62, 0,58,25, 6,42,49,17,54, 5,24,78,68,57,38,39,47,
33,63,57,10,18,21, 7,41,34,71,51,68,70,52, 0, 4,13,62,69,30,15,14,67,35,29, 1,78,54, 5,24,64, 3,60,65,20,48,50,45,31,17,79,39,38,
   28,11,36,49,12,72,66,77,44, 6,40,59,61,53,22,37,16,43,19, 9,55,46,74,73,26, 8,23,75, 2,32,27,58,47,56,25,42,76,
72,69,52, 4,59,22,14, 3,70,26,61,36,63,29,53,79,49,15,62,33, 9,66, 0,16,44,45,58,41,43,24,71,27,23,67,21,25,32,46,39,55,
   64,50,35,57,60,13,19,11, 1,12,37, 5,56,10,20,75,30,68, 6,28,47,65,34,40, 8,76,78,48,77, 7, 2,74,51,73,38,17,42,31,54,18,
15,20,40, 9,61,45,32,63,28,64,37,34, 4, 3,65,27,25,66,30, 5,24,29,70,26,59,22,77,54,78,11,19,60,33, 8,55,10,31,46,13,16,69,
   51,73,44,39,53, 1,17,72,48,57,74,56,43,36, 2,76,68,38,14,79,52, 0,21,12,49,18,50,41, 6,42, 7,62,71,75,47,58,35,67,23,
43,40, 9,64,62,17, 3,72,22,52,20,63,29,37, 1,74,79,76,57,59,77, 2,33,25,58,12,75,47,31,26, 8,38,27,45,55, 6,19,67,69,48,61,23,32,
   5,15,66,14,30,36, 7,24,34,71,42,73,49, 0,39,51,70,78, 4,41,56,28,50,21,18,35,10,68,60,44,53,46,11,16,13,65,54,
57,34,72,75,79, 8,53,60,43,64,41,35,25,16,24,70, 5, 4,76,46,44,74,45,32, 1,49,39,37,19,21,65,68,78,27,47, 2,22,11, 0,20,13,12,18,
   3,66,42,36,62,17,54,29,71,26,56, 6,40,38,50,61,73,77,59,67,63,23,10,15,14,28,33,69,55, 9,48,58,30,51,31,52, 7,
48,30,19,59,52,26, 2,37,22,13,40,42, 4,72,63,28,71,56,21,73,79,15,50,20,64,58,70,47,53,65,27, 7,25,76, 5,61,45,67, 9,
   0,18,60,31,29,14, 6,17, 3,57,68,24,43,34,66,49,33,36,55,35,69,46,62,16,39,78, 1,32,75,51,11,12,10,77,23,54, 8,74,44,38,41,
```

```
29, 3,54,68,16, 1,55,32,69,67,19,75,37,46,71,33,62,30,57,27,58,53,42,52,38,28,56,73,63,24,25,39, 8,79,18, 2,34,36,10,12,
   15,11,50,49,51,22,14,70,66,23,31,44,61,74,17,76, 7,77,48,43, 5, 4,65,35,78, 9,13, 6,45,26,40,41,64,47,20,72,21,59, 0,60,
19,44,49,56, 5,14,40,63,66, 0,11,51,25,13,78, 3,74,20, 6,70,64,65, 9,23,37,67,36,38, 8,59,60,18,21, 2,42,35,53,43,75,62,
   39,33,12,79,73,68,32,30,16, 7,57,31,41,26,71,22,17,24,47,10,15,34,61,55,46,72,54,77,50, 1,29,27,76,45,52,69,28,58,48, 4,
16,13, 0,55,40,42,15, 1,23,32,47,63,68,73,52,74,10,78,21, 8,43,66,39,11,29,37,45,53,27,36,41,49,54,12,69,76, 2,64,19,60,
   3,44,34,79,22, 6,20,58,14,56,71, 4,72,70,25,48,57,67,46,65,31, 5,75,33,38,18,30,35, 9,59,62,17, 7,28,61,50,77,51,24,26,
39,18,56,32,11,54,73,72,21,24,41,14,76, 7, 9,70,71,12,58,38,57,50,67,51,28,26,79,77,33,53,35,63, 8,37,68,64,75,20,62,16,55, 2,10,
   4,42,69,22,13,43,48,52,46, 0,23,60, 6,15,40,17,36,29, 3,31,19,25,27,44,49,59, 5,65,66,45,30, 1,47,34,74,61,78,
7,23,74,78,56,53, 0,54,20, 9,50,73, 6,35,45,33,42,52,18, 1,65,72,55,79,77,58,17,34,22,30,36,70,59,26,57,21,71,44,43,28,75,10,13,
   8,63,64, 4, 5,27,39,31,62,25,60,48, 3,12,11,14,66,16,37,40,38, 2,46,67,24,51,49,15,69,68,76,41,47,19,61,32,29,
44,42,17,12,66,11, 0,69,49,67,53,65,61, 9,72,10,18,68,25,58,22,75,51,16,64,19,79,55,27,15,77,23,54,47, 7,50,60,41,57,73,34,56,
   2,21,62, 6,26, 5,13,40, 1, 4,29,38,33,31,78,39,46,37,24,63,20,74,59,36,45, 3,35,70,30,43,71,32,14, 8,28,48,76,52,
48,77,61,65, 8,23,55,37,21,28,36,18,26,24, 1,43,57, 9, 4,53,78,67,79,63,50,72,29,58,59,20,60,46, 0,44,70,30,15,
   5,49,73,54,40,71,64,10,69, 3,56,17,41,76,42,34,32,19,27,47,62, 7,25,51,68,31,74,13,14,38,35,75, 2,22,45, 6,39,11,16,33,66,12,52,
38,77,71,48,20,52,25,36,27,79,22,16, 2,18,32,46,39,59,73, 1,75,78,57,44,42, 4,10, 9,58,60,61,17,15,50,51,31,35,54,63,62,
   72,65,19,53,14,56,34,26, 6,67,24,70,66,43,29,23,76,49,37, 7,74,28,12,13, 8,40,47,30,11, 5,69,45, 3,68, 0,64,21,41,33,55,
73,57,21,48,53,12,36,17,58,78,75, 7,50,64, 8, 0,29,77,22,55,54,61,38,59,70,24,68,13, 6,11,35,41,44,45,52,76,23,60,39,
   9,67,18,43,66, 2,10,72,28,47,15,62,42,56,51,33,65,74, 5,69,40,25,20, 1,34,16,27, 3,37,19,14,26,32,79, 4,30,46,49,31,71,63,
58,29,10,41,51,59, 0,13, 5,63, 4,37, 8, 3,61,54,79,47,67,23,48,77,52,71, 9, 6,11,68,25,60,15,62,65,32,21,70,73,55,46,22,56,45,
   40,38,64,33,75,18,57,69,53,76,16,42,35,19, 7,34,74,78,44,72,14, 2,17,24,12,27,26,50,66,36,20,43,49,31, 1,30,39,28,
73,36,69,25,66,45,11,29,27,42,23,10,22, 5, 3, 6,38,50,75, 7,55,43,79,16,47,63,48,68,72,58, 9,67,60,40,37,35,14,15, 4,46,52,65,
   0,53,18,32,19,64,17,44,77, 8,57,74,21,70,20,28,62,54,39,12,61,13,26,30,49,41,59, 1,78,76,34,71,24,33,56,31, 2,51,
22,52,71,57,31, 1,36,50,28,63,30,21, 3,13,16,10,58, 5,35,23,29,60,20,73,24,79,75, 8,51,66,26,62,43,45,78,27,49,25,41,
   0,11,38,67,14, 2,61,55,46,53,64,42,47,59, 6,65,40,39, 9, 4,54,70,69,68,19,72,17,34,15, 7,18,37,74,76,32,12,44,33,56,77,48,
29,44,76, 9,30,25,60,35,45, 5,22,15,40,66,34,59,18,13,58, 3,33,65,14,36,75,72,39,20,69,55,38,79,26,17,50,48,54,78,
   52,31,43,27,32,68, 2,67,42, 1,71,46, 4,41,53,37,74,12, 6,57,28,16,56,23,64,19, 7,10,47,21,63,70,62,11,24, 0,49,77, 8,61,51,73,
25,38,27,56,65,49,42,28,10,76,37,34,74,57,59,67, 8,47,26,15,33,23,40,71,50, 2,46,21,19, 6, 4, 5, 9,43,18,63,45,73,58,53,14,64, 7,
   0,48,20,54,68, 1,52,60,77,22,72,62,51,30,78,11,61,39,36,66,31,32,17,35,79, 3,24,69,13,55,16,44,75,70,41,29,12,
45,71,66,11,54,56, 3,51,35, 0,53,19,21,55,10, 2,73,27,59, 1,31,60,20,62,68,22,38,74,61,50,58,23,24,49,48,28,12,13,
   6,44,43,37,25,26,29,75,36,46,32,70,16,76,39,17,30,41,47,67,40,15,78,14, 9, 8,69,63,79,72,57,33, 4,42,34,52,65,77, 7,18, 5,64,
16,55,63,42,48,27,38,28,57,72,62,19,79,33,15,77,51, 1,36, 7,78,52,29,25, 2,59,18,12,39,64,46, 3,24,43,60,34,61,40,54,14,44,74,
   5,47,75, 9,49,23,68,58,65,26,56,22,37,20,66,21, 6,71, 0,17,73,50,41,10,35, 8, 4,70,30,67,69,76,53,32,13,45,31,11,
49,60, 4,13,30,20,78, 3,37, 5,50,43,73,75, 6,40,23,29,11,53,27,31,34,74,64,41,36,17,44,48,77,21,16,18,58,79,57,10,
   7,22,54,52,35,25,26,33,28,71,12,69,63,65,24,59,47,76,15,56,72, 8,46,39,42,45,55,38,66,14,67,62, 2,51,68, 1,70,32,19, 0, 9,61,
19, 3,75,59,10, 8,14,11,12,39,67,41,28,74,76,73, 7,33,35,55,65,15,77,49,24,37,13,44,30,45,47, 4,70,36,50,69, 5,62,34,22, 0,61,
   1,71,42,54,20,60,40,53,57,26,72,46,31,63,52,18,17,29,21, 9,58,66,51, 6,38,23,16,64,32,27,79, 2,68,48,78,56,43,25,
27,71,73,66,12,47,44,63,33,11,61,72,46,69,31,48, 3,16,65,24,40,49,77, 1,58,70,14,52,57,21, 8,64,13,59,10,55,23,
   0,17,53,41,54,68,78,67,38,39, 9,51,76,45, 7,62,60, 4,22,15,37,29,25,34,26,28,79, 5,20,74,18,56,32,42,43, 6,19,75,50,30,35,36, 2,
27,20,10,54, 8,57,40, 0,22,12,47,36,75, 7,35,45,19,34,72,58,74,23,16,33,64,14,78,39,59,24,11,26, 6,28,32,43,73,38,67,25,70,71,
   42,66, 3,46, 9,60,15, 2,51,21,79,53,30,65,41,68,13,56,76,77, 4, 5,37,44,49,52,63,17,29,31,50,61,69,62,55,48, 1,18,
56,59,62,67,26,44,10,18,57,55,70,63,78,73, 0,40,34,29,30,42,11,68,72,64,12,35,50, 9,39,17,27,61,24, 2,31,47,41,53, 5,33,49,54,
   1,45,15,25,77, 8,20,75,60,16,32,36,28,52,38,43,74,71,37,21,13,66,79,65, 7,19,48,23,22, 3,46,58,76,14,69,51, 4, 6,
8,56,60,48,53, 2,40,66,29,10,61,59,11,67,35,51,14,36,63,76,54,44,69, 0, 3,15,32,79,73,58,72,23,12,71,57,50,19,52, 7,75,77,45,70,33,
   1,55,68,22,42,49,38, 4,47,24,30,17,37,27,43,39,78,34,65,28,13,64, 9,25, 6,20,46,74,62,31, 5,41,16,21,26,18,
```

```
40,73,77,48,59,37, 7,56,60,58,79,13, 6,21,30,23,47, 9,42,36,74,72,66, 3,38,25,45, 0,39,67,34,31,61,29,33,75,27, 1,68,15,41,
   5,50,78,69,63,76,71,51,44,52,12,54,19,32, 8,11,64, 4,28,65,26,49, 2,55,46,43,62,35,17,70,22,20,57,14,24,10,16,18,53,
34,16,44,30,25,78,29, 0,71,54,49,47,14,26, 2,20,22,61,53,36,59,27,38,42,43,52,74,39,24, 4,70,63,45,46,64,56, 1,66,35,
   5,68,51,65,72,28,19, 7,48,18,13,33,21,60,50,40,76,37,55,32,79,77,69, 3,23,75,58, 6,41,62,10,67,15, 8,11,73,12,57, 9,17,31,
35,34,68,54,11,13,41,33,52,16,63,72,60,46,48,69,28,40,12,73, 2, 0,29,53,37, 9, 5, 4,14,78,39,50,47,70,45,64, 7,25,79,18,21,
   1,36,67,55,42,20, 6,76,26,59,51,19, 8,24,65,43,44,49,57,56,17,38,61,27,77,32,66,30,74,23,10,15,75,31,58,62,71,22, 3,
35,16,41,21,28,79, 3,32,27, 2,51,48,31,44,18,39,13,74, 6,59,76,58, 8, 9,14,68,63,62,24, 0,15,22,49,23,47,65,33,78,70,56,66,
   1,42,10,17,38,40,67,64,69,12,19,71,36,11,52,50,72,77, 7,53, 4,26,46,43,37,34,73,20,45,57,54,55,30, 5,25,60,75,61,29,
1,30,52,57,60,34,67,10,72, 5,19,27,21,44, 4,74,61,47,64,39,40,28,45,26, 6,42,53, 8, 3,55,37,36,73,24,15, 0,56,13,77,14,16,25,
   62,51,70,79,22,18,43,12,17, 9,29,65,59,68,63,31,48,32,46,33,58,35,38,41,11,66, 7,69,50,23,75,71, 2,54,49,76,78,20,
18,34, 1,31,14,40,79,74,78,63,70,19,17,12,62,69, 9,48,24,77,33,76,20,21,55,64,66, 8,51,26,25, 2,73,60,49,75,43,29,54,
   28,53,23,38,45,50,52,35, 5, 4,22,57,61,68,27,11, 3,32,72,56, 7,41,39,30,36,42,65,58,59,37,47, 6,15, 0,13,16,46,44,10,67,71,
68,69,58,72,26,42,32,38,31,70,67,64,55,13,29,59,33,78,14,66,11,28,48,44,36,75,35,76, 5,54,77, 8,30, 0,37,62,73,21,63,25,34,12,
   6,23,60,27,53,40,52,56,46, 7, 1,39, 3,41,24,10,20,19,18,57,71,15,51,16,47,43,17,74, 2,65, 9,45,61,22,79,50, 4,49,
```

B.4 Zpool status for the 80 drive JBOD

The following is the complete ‘zpool status’ listing for the 80 drive dRAID created in Section A.1.1. The array has 3 distributed spare drives and 7 (8+3) parity groups.

```
# zpool status
pool: MS09
 state: ONLINE
 scan: none requested
config:
NAME STATE READ WRITE CKSUM
 MS09 ONLINE 0 0 0
  draid3-0 ONLINE 0 0 0
    sdb    ONLINE 0 0 0
    sdd    ONLINE 0 0 0
    sde    ONLINE 0 0 0
    sdg    ONLINE 0 0 0
    sdh    ONLINE 0 0 0
    sdi    ONLINE 0 0 0
    sdk    ONLINE 0 0 0
    sdl    ONLINE 0 0 0
    sdm    ONLINE 0 0 0
    sdo    ONLINE 0 0 0
    sdp    ONLINE 0 0 0
    sdq    ONLINE 0 0 0
    sds    ONLINE 0 0 0
    sdt    ONLINE 0 0 0
    sdu    ONLINE 0 0 0
    sdw    ONLINE 0 0 0
    sdx    ONLINE 0 0 0
    sdy    ONLINE 0 0 0
    sdz    ONLINE 0 0 0
    sdab   ONLINE 0 0 0
    sdac   ONLINE 0 0 0
    sdad   ONLINE 0 0 0
    sdae   ONLINE 0 0 0
    sdc    ONLINE 0 0 0
    sdf    ONLINE 0 0 0
    sdj    ONLINE 0 0 0
    sdn    ONLINE 0 0 0
    sdr    ONLINE 0 0 0
    sdv    ONLINE 0 0 0
    sdaa   ONLINE 0 0 0
    sdaf   ONLINE 0 0 0
    sdag   ONLINE 0 0 0
    sdah   ONLINE 0 0 0
    sdai   ONLINE 0 0 0
    sdaj   ONLINE 0 0 0
    sdak   ONLINE 0 0 0
    sdal   ONLINE 0 0 0
    sdam   ONLINE 0 0 0
    sdan   ONLINE 0 0 0
    sdao   ONLINE 0 0 0
    sdap   ONLINE 0 0 0
    sdaq   ONLINE 0 0 0
    sdar   ONLINE 0 0 0
    sdas   ONLINE 0 0 0
    sdat   ONLINE 0 0 0
    sdau   ONLINE 0 0 0
    sdav   ONLINE 0 0 0
```

```
     sdaw   ONLINE 0 0 0
     sdax   ONLINE 0 0 0
     sday   ONLINE 0 0 0
     sdaz   ONLINE 0 0 0
     sdba   ONLINE 0 0 0
     sdbb   ONLINE 0 0 0
     sdbc   ONLINE 0 0 0
     sdbd   ONLINE 0 0 0
     sdbe   ONLINE 0 0 0
     sdbf   ONLINE 0 0 0
     sdbg   ONLINE 0 0 0
     sdbh   ONLINE 0 0 0
     sdbi   ONLINE 0 0 0
     sdbj   ONLINE 0 0 0
     sdbk   ONLINE 0 0 0
     sdbl   ONLINE 0 0 0
     sdbm   ONLINE 0 0 0
     sdbn   ONLINE 0 0 0
     sdbo   ONLINE 0 0 0
     sdbp   ONLINE 0 0 0
     sdbq   ONLINE 0 0 0
     sdbr   ONLINE 0 0 0
     sdo    ONLINE 0 0 0
     sdp    ONLINE 0 0 0
     sdq    ONLINE 0 0 0
     sds    ONLINE 0 0 0
     sdt    ONLINE 0 0 0
     sdu    ONLINE 0 0 0
     sdw    ONLINE 0 0 0
     sdx    ONLINE 0 0 0
     sdy    ONLINE 0 0 0
     sdz    ONLINE 0 0 0
     sdab   ONLINE 0 0 0
     sdac   ONLINE 0 0 0
     sdad   ONLINE 0 0 0
     sdae   ONLINE 0 0 0
     sdc    ONLINE 0 0 0
     sdf    ONLINE 0 0 0
     sdj    ONLINE 0 0 0
     sdn    ONLINE 0 0 0
     sdr    ONLINE 0 0 0
     sdv    ONLINE 0 0 0
     sdaa   ONLINE 0 0 0
     sdaf   ONLINE 0 0 0
     sdag   ONLINE 0 0 0
     sdah   ONLINE 0 0 0
     sdai   ONLINE 0 0 0
     sdaj   ONLINE 0 0 0
     sdak   ONLINE 0 0 0
     sdal   ONLINE 0 0 0
     sdam   ONLINE 0 0 0
     sdan   ONLINE 0 0 0
     sdao   ONLINE 0 0 0
```

```
      sdap   ONLINE 0 0 0
      sdaq   ONLINE 0 0 0
      sdar   ONLINE 0 0 0
      sdas   ONLINE 0 0 0
      sdat   ONLINE 0 0 0
      sdau   ONLINE 0 0 0
      sdav   ONLINE 0 0 0
      sdaw   ONLINE 0 0 0
      sdax   ONLINE 0 0 0
      sday   ONLINE 0 0 0
      sdaz   ONLINE 0 0 0
      sdba   ONLINE 0 0 0
      sdbb   ONLINE 0 0 0
      sdbc   ONLINE 0 0 0
      sdbd   ONLINE 0 0 0
      sdbe   ONLINE 0 0 0
      sdbf   ONLINE 0 0 0
      sdbg   ONLINE 0 0 0
      sdbh   ONLINE 0 0 0
      sdbi   ONLINE 0 0 0
      sdbj   ONLINE 0 0 0
      sdbk   ONLINE 0 0 0
      sdbl   ONLINE 0 0 0
      sdbm   ONLINE 0 0 0
      sdbn   ONLINE 0 0 0
      sdbo   ONLINE 0 0 0
      sdbp   ONLINE 0 0 0
      sdbq   ONLINE 0 0 0
      sdbr   ONLINE 0 0 0
      sdbs   ONLINE 0 0 0
      sdbt   ONLINE 0 0 0
      sdbu   ONLINE 0 0 0
      sdbv   ONLINE 0 0 0
      sdbw   ONLINE 0 0 0
      sdbx   ONLINE 0 0 0
      sdby   ONLINE 0 0 0
      sdbz   ONLINE 0 0 0
      sdca   ONLINE 0 0 0
      sdcb   ONLINE 0 0 0
      sdcd   ONLINE 0 0 0
    spares
      $draid3-0-s0 AVAIL
      $draid3-0-s1 AVAIL
      $draid3-0-s2 AVAIL

errors: No known data errors
```

Appendix C. References

[1] I. Huang, "Declustered RAIDZ Scope Statement," Argonne Contract number: B609815, MS6, 2015.

[2] I. Huang, "Declustered RAIDZ Solution Architecture," Argonne Contract number: B609815, MS6, 2015.

[3] G. Alvarez, W. Burkhard, L. Stockmeyer and F. Cristian, "Declustered Disk Array Architectures with Optimal and Near-optimal Parallelism," in *Proceedings of the 25th International Symposium on Computer Architecture, ISCA '98*, 1998.

[4] M. Holland and G. Gibson, "Parity Declustering for Continuous Operation in Redundant Disk Arrays," in *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, 1992.

[5] E. Riedel, "http://www.cs.cmu.edu/~riedel/ftp/Declustering/BD_database.tar.Z," [Online].

[6] T. Scharz, J. Steinberg and W. Burkhard, "Permutation Development Data Layout (PDDL)," in *Proc 5th IEEE Symp. on High Performance Computer Architecture, HPCA'99*, 1999.

[7] A. Brinkmann, K. Salzwedel and C. Scheideler, "Efficient, distributed data placement strategies for storage area networks (extended abstract)," in *Proceedings of the Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '00*, 2000.

[8] J. R. Santos, R. R. Muntz and B. Ribeiro-Neto, "Comparing random data allocation and data striping in multimedia servers," *SIGMETRICS Perform. Eval. Rev.,* vol. 28, pp. 44-55, 2000.

[9] A. Goel, C. Shahabi, S.-Y. D. Yao and R. Zimmermann, "SCADDAR: An efficient randomized technique to reorganize continuous media blocks," in *Proceedings of the 18th International Conference on Data Engineering, ICDE '02*, 2002.

[10] B. Seo and R. Zimmermann, "Efficient disk replacement and data migration algorithms for large disk subsystems," *Trans. Storage,* vol. 1, pp. 316-345, 2005.

[11] D. Brady, "Lustre Sreaming Improvements High Level Design," Argonne Contract number: B609815, MS3, 2015.

[12] "Knuth Shuffle," [Online]. Available: https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle. [Accessed 2015].