OpenZFS

dRAID, Finally!

•••

Mark Maybee

Hewlett Packard
Enterprise

Topics

OpenZFS

- What is dRAID?
- Issues and Answers
  - Permutation arrays
  - Group size
  - Stranded space
  - Space inflation
- Drive Replacement Performance
- When is dRAID going to be available?
- Next steps?

History

OpenZFS

This feature was originally developed at Intel by Isaac Huang (see his OpenZFS Summit talks in 2015, 2016, and 2017). It languished for a time before being rebased by Don Brady. The feature was “picked up” by Cray in 2018 to be used as a key capability in the ZFS version of its distributed storage product. Brian Behlendorf adopted the Cray version of the feature in 2020 and created the current PR. He has worked with Mark Maybee from Cray to keep it rebased and address a number of outstanding issues which needed to be addressed prior to integration.

Terminology

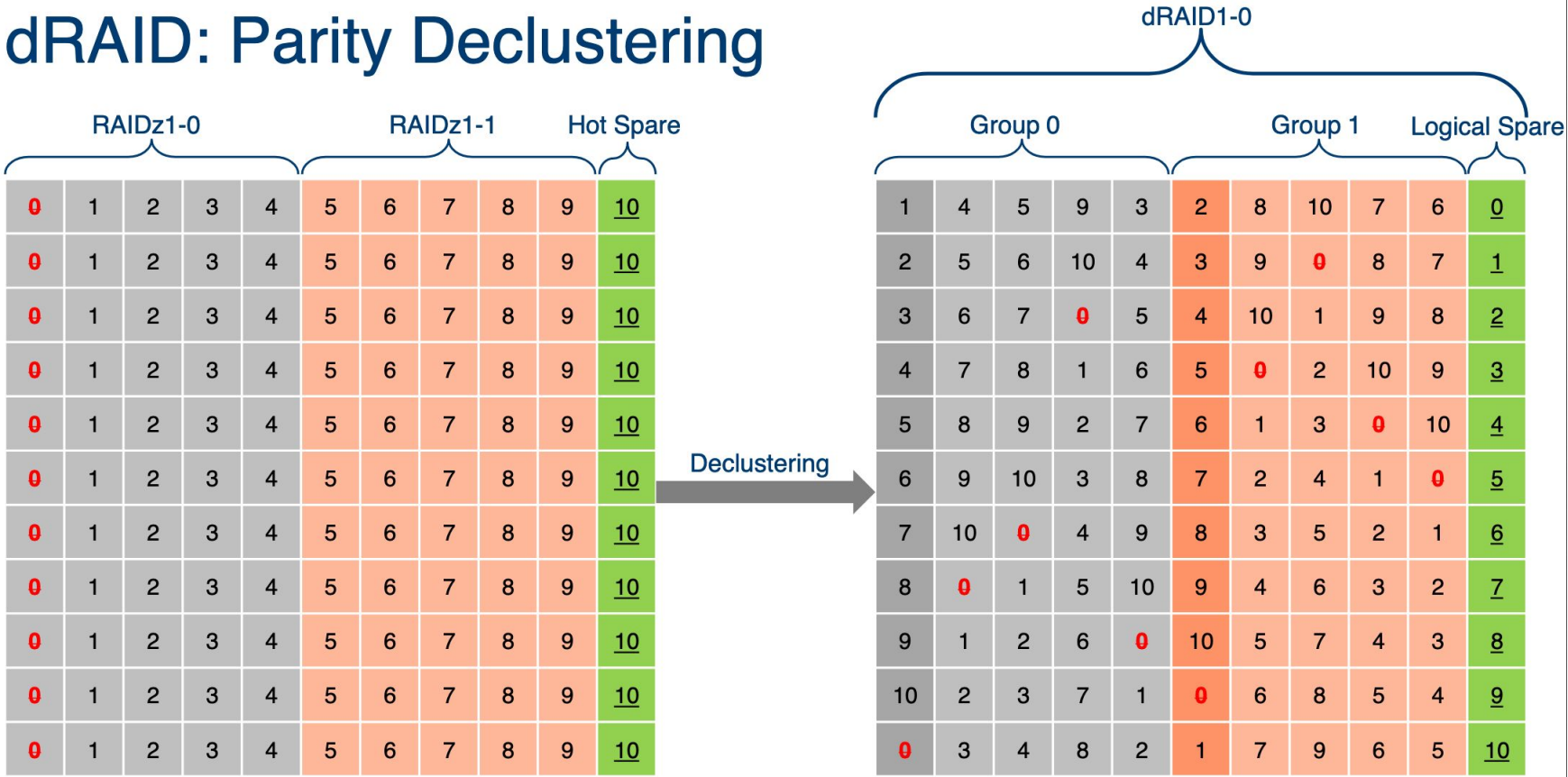
OpenZFS

- Group Size - the number of pieces the data is partitioned into plus the amount of parity
  - The amount of parity determines the redundancy
  - The number of data pieces determines the overhead
- dRAID Size - the number of drives used for data
  - Does not include spare drives
- dRaid Row - a 16M chunk from each drive at the same offset
  - E.g., Row 0 starts at offset 0 of each drive
- Permutation Slice - 1 or more rows which are permuted based on the dRAID permutation array
  - Number of rows per slice is derived from LCM of Group Size and dRAID Size

What is dRAID?

OpenZFS

dRAID: Parity Declustering



dRAID vs RAIDz

OpenZFS

- dRAID implementation is layered on RAIDz code
  - Different map_alloc function, same IO pipeline
- RAIDz layout
  - Each group is constructed from a set of physical drives
  - Group columns extend full length of drive
  - Allocations must be multiple of parity + 1
  - Example: 1K data requires 2K space with 1 parity
- dRAID layout
  - Groups divided into “rows” (16MB)
  - Group rows are non-contiguous on physical drives
  - Allocations must be multiple of parity + data
  - Example: 1K data requires 2.5K space in 5 disk group

RAIDz1

Disk / LBA	A	B	C	D	E
0	P_0	D_0	D_1	D_2	D_3
1	P_1	D_4	D_5	D_6	D_7
2	P_0	D_0	D_1	D_2	P_0
3	D_0	P_0	D_0	D_1	X

dRAID1

Disk / LBA	A	B	C	D	E
0	P_0	D_0	D_1	D_2	D_3
1	P_1	D_4	D_5	D_6	D_7
2	P_0	D_0	D_1	D_2	X
3	P_0	D_0	X	X	X
4	P_0	D_0	D_1	X	X

Why Decluster?

OpenZFS

- Spare drives are leveraged
  - Spare capacity is distributed across all spindles
- Each group's data is randomly distributed using all drives
  - Decouples redundancy group from number of data drives
- Data reconstruction leverages all drives
  - Reads shared evenly among all drives
  - Writes go to spare blocks distributed across all drives
- Sequential Resilver works!
  - Any row can be reconstructed without block pointer data

draid1-0

4	8	2	16	1	.	.	.	.	.	29	27	30	23	15	28	25	14	19	7	11	22	21	13	26
5	9	3	17	2	.	.	.	.	.	30	28	0	24	16	29	26	15	20	8	12	23	22	14	27
6	10	4	18	3	.	.	.	.	.	0	29	1	25	17	30	27	16	21	9	13	24	23	15	28
7	11	5	19	4	.	.	.	.	.	1	30	2	26	18	0	28	17	22	10	14	25	24	16	29
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

distributed spare

0
1
2
3
.

Creating a dRAID

OpenZFS

- New top-level vdev type: draid
  - Similar to raidz
  - Draid1, draid2, and draid3 supported
- Specify redundancy group size
  - Ex: draid1:5d -> 5+1 group size
  - Defaults to 8
  - Must be less than number of data drives in draid config
- Specify spare capacity (drive count)
  - Ex: driad1:5d:2s -> 2 drives worth of spare
- Config exposed in zpool status output
  - Spare capacity exposed as pseudo drives

```
# zpool create tank draid2:4d:2s L0 … L52
# zpool status
 pool: tank
 state: ONLINE
config:

    NAME                    STATE     READ WRITE CKSUM
    tank                    ONLINE       0     0     0
      draid2:4d:53c:2s-0    ONLINE       0     0     0
        L0                  ONLINE       0     0     0
        L1                  ONLINE       0     0     0
        L2                  ONLINE       0     0     0
        L3                  ONLINE       0     0     0
        ...
        L50                 ONLINE       0     0     0
        L51                 ONLINE       0     0     0
        L52                 ONLINE       0     0     0
    spares
      draid2-0-0    AVAIL
      draid2-0-1    AVAIL

errors: No known data errors
```

Permutation Array

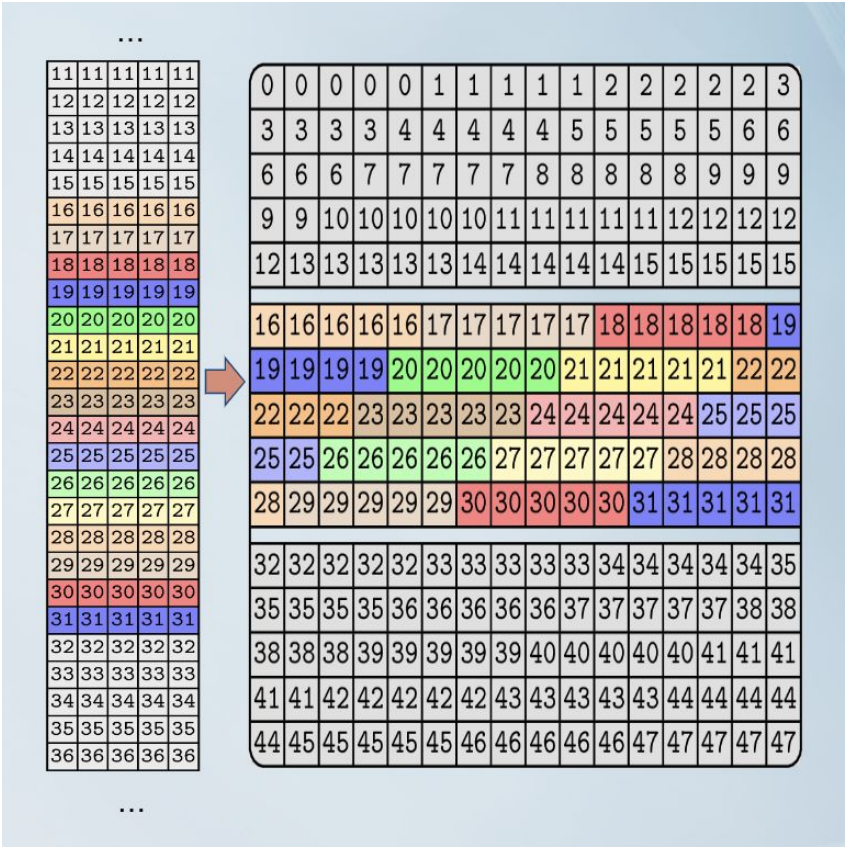
OpenZFS

- Need to provide the drive permutation for each slice
- Generated at pool creation
- If lost, all data access is lost
- Stored in the label
  - Takes up lots of space
- Answer: Pre-define the permutations
  - No need to figure it out at pool creation
  - No need to store in label
  - No risk of “losing” this information

Group Size Constraints

OpenZFS

- Constrained by draid “size”
  - Size = number of drives - number of spares
  - Group size must divide evenly into draid size
  - E.g., 30 drives can only support
    - 3 drive group
    - 5 drive group
    - 10 drive group
    - 15 drive group
- Answer: Tiles
  - Use multiple rows for each dRAID slice
  - Only need to specify group size at creation
  - Number of groups in a slice = LCM of group size and dRAID size



Stranded Space

OpenZFS

- Block allocation must fit within a group slice
  - Group slice size = group size * 16M
  - If block does not fit into space available in group slice it is allocated in next available group slice
  - Space can remain unallocated at end of group slice
  - There are many group slices in a dRAID
- Answer: Blocks spanning groups
  - Now can split a block across two groups
  - Leverage multi-row allocation maps developed for RAIDz expansion project

Space Inflation

OpenZFS

- Unlike RAIDz, all columns are filled in an allocation
  - Particularly significant with small blocks
  - Any block that is not a multiple of *group size * sector size* will end with pad sectors
  - Pad sectors are explicitly zero-filled for sequential resilver to work

RAIDz1

LBA \ Disk	A	B	C	D	E
0	P_0	D_0	D_1	D_2	D_3
1	P_1	D_4	D_5	D_6	D_7
2	P_0	D_0	D_1	D_2	P_0
3	D_0	P_0	D_0	D_1	X

dRAID1

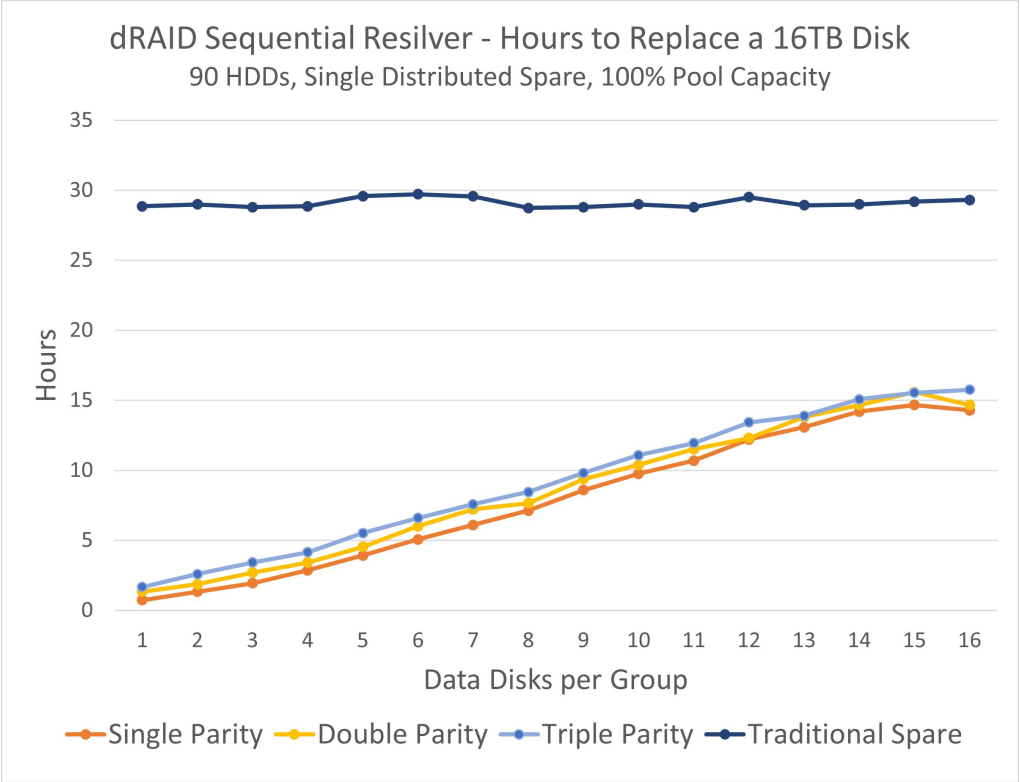
LBA \ Disk	A	B	C	D	E
0	P_0	D_0	D_1	D_2	D_3
1	P_1	D_4	D_5	D_6	D_7
2	P_0	D_0	D_1	D_2	X
3	P_0	D_0	X	X	X
4	P_0	D_0	D_1	X	X

- Answers: Large blocks & Allocation classes

Drive Replacement Performance

OpenZFS

- Uses distributed spare
  - Spare capacity on all drives
- Declustering distributes reads
  - Failed drive impacts all groups
  - Reconstructed using all groups
- Smaller groups == faster rebuild
  - Fewer reads for reconstruction
- Much faster than “hot spare”
  - Small groups 10x faster
  - Large groups 2x faster



When will dRAID be Available?

OpenZFS

- Originally targeting OpenZFS 2.0 release
  - Hence the title of this talk
  - Last minute changes/improvements pushed us out
- Now targeting the OpenZFS 2.1 release
  - Late 2020, early 2021 time frame

Next Steps

OpenZFS

- Get it integrated!
  - Some small test case issues
  - Final code reviews
- dRAID expansion?
  - See you at the code-a-thon!

Questions?

Performance Tease

OpenZFS

- Preliminary performance results
- *Includes DirectIO support*
- 12 NVMe drives
  - dRAID2:d9:s1 configuration
    - 1 group of 9+2, 1 spare
  - 30G/s hardware write bandwidth
  - 42G/s hardware read bandwidth
- Achieving 80% of max write
  - 24G/s streaming writes
- Achieving 85% of max read
  - 36G/s streaming reads