SEE TEXT ONLY

Monitoring ZFS

BY ALLAN JUDE

ZFS is an advanced filesystem, but it is also one of the most observable. The combination of static and dynamic DTrace probes, statistics, and tooling built into ZFS means it is one of the easiest filesystems to do performance analysis on. This article covers checking the health of the pool, measuring load and performance in real time, and using historical statistics to detect changes in health, performance, or behavior. Then provides a brief overview of other tools to consider for monitoring a system in even more detail. First up, is the pool healthy?

HEALTH CHECKING • zpool Status

The zpool status command is the first place to look to assess the general health of the pool. It prints a visual representation of the layout of the devices in the pool, and the status of each device. In addition to the status of each device, there are also columns of counters, showing the number of read, write, and checksum errors that have been detected on each device.

Each device can be in one of these states:

- **Online** — The device is healthy and working as expected.
- **Offline** — An administrator has marked this device as offline.
- **Degraded** — The device is functioning at a reduced capacity. Usually only applies to a top level vdev, like RAID-Z or a Mirror; indicates that one or more member disks has failed and the system is using parity to remain operational.
- **Faulted** — The device or pool is no longer working because too much data is missing. If a device or pool loses more devices than it has redundancy, files may be inaccessible or lost. Try to reconnect the missing devices to continue.
- **Removed** — An underlying device has been removed. This can happen when a disk fails and the device is removed by the operating system, or when a disk is physically disconnected by an operator.
- **Missing** — ZFS was unable to find, or unable to open, the device. Try to reconnect the device, or solve the error that prevented ZFS from opening the device (such as it being in use by another process). The zpool online command is useful to bring a device back online.
- **Replacing** — A device is being replaced. When replacing a device that is online, a new top-level vdev called replacing-X (where X is an incrementing integer) will be created; it is effectively a mirror with the new and old devices as members. Data is copied from the old device to the new device. Once the operation is complete, the old device will

be detached from the pool, and the new device will become a regular member of the vdev.

- Spare — The device is missing or otherwise degraded and has been temporarily replaced with a spare.
- Resilvering — This device was temporarily offline or has suffered some corruption and the missing or damaged data is being replaced from available parity.

```
# zpool status
  pool: media
 state: ONLINE
  scan: resilvered 25.7M in 0h0m with 0 errors on Sat Oct 14 14:40:18 2017
config:

        NAME                        STATE     READ WRITE CKSUM
        media                       ONLINE       0     0     0
          raidz2-0                  ONLINE       0     0     0
            gpt/s5-Z5009MV3         ONLINE       0     0     0
            gpt/s3-Z500Z78C         ONLINE       0     0     0
            gpt/s2-Z500ZKL8         ONLINE       0     0     0
            gpt/s4-Z503E2PR         ONLINE       0     0     0
            gpt/s1-Z1F3134B         ONLINE       0     0     0
            gpt/s6-Z500XXPA         ONLINE       0     0     0

errors: No known data errors
```

If one or more problems are detected with the pool, a summary will be displayed at the end of the status output. Running zpool status -v <optional poolname> will extend this summary, and provide a list of each file that has suffered damage, allowing those individual files to be restored from backups.

```
# zpool status -v zroot
  pool: zroot
 state: DEGRADED
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://illumos.org/msg/ZFS-8000-8A
  scan: resilvered 1.43T in 79h21m with 3 errors on Sat Oct 14 01:18:56 2017
config:
       NAME                           STATE     READ WRITE CKSUM
       zroot                          DEGRADED   113     0     0
         raidz1-0                     DEGRADED   113     0     0
           ada0p3                     ONLINE       0     0     0
           ada1p3                     ONLINE       0     0     0
           ada2p3                     ONLINE     113     0     0
           replacing-3                OFFLINE      0     0     0
             17161359962879308376     OFFLINE      0     0     0
             ada3p3                   ONLINE       0     0     0


errors: Permanent errors have been detected in the following files:

        /usr/src/contrib/binutils/ld/emultempl/armcoff.em
        /usr/src/contrib/binutils/ld/emultempl/armelf.em
        /usr/src/contrib/binutils/ld/emultempl/astring.sed
        /usr/src/contrib/binutils/opcodes/ChangeLog-2006
```

The zpool status command also tracks the progress of scrub and resilver operations, including an average speed and a completion estimate. The speed estimate is an average for the entire operation, which is much slower for the first few %, so consider waiting until 5%–10% completion before taking the speed and ETA seriously. If the system is rebooted or otherwise interrupted during the resilver operation, the estimate may be stuck at 1 byte per second for a long time.

```
# zpool status -v zroot
  pool: zroot
 state: ONLINE
  scan: scrub in progress since Wed Oct 18 22:27:19 2017
        20.7G scanned out of 32.1G at 401M/s, 0h0m to go
        0 repaired, 64.50% done
config:

        NAME                     STATE     READ WRITE CKSUM
        zroot                    ONLINE       0     0     0
          mirror-0               ONLINE       0     0     0
            gpt/i1-14450DAFF1A8  ONLINE       0     0     0
            gpt/i2-154310EB96A5  ONLINE       0     0     0
```

S.M.A.R.T.

Disks also provide some insight into the state of their health using the SMART (Self-Monitoring, Analysis and Reporting Technology) protocol. For spinning disks, the greatest indicators of impending failure are usually the number of pending and reallocated sectors. Each manufacturer provides a different set of statistics, so it is difficult to create hard and fast rules about what means the disk is underperforming or indicating potential failure. To make the most sense out of the various counters in the SMART status, you need a reference point, what those counters looked like in the past, how much and how fast they have changed. Another condition to watch out for is disks that have a high and rapidly growing cycle count. If the disk is going to sleep and waking up every few seconds, this will put tremendous wear on the disk. The disk may need to be given specific commands to adjust the idle timeout, or need updated firmware to fix the problem. SAS disks generally provide fewer counters but are more consistent across drive models and manufacturers.

```
# smartctl -a /dev/ada1
=== START OF INFORMATION SECTION ===
Model Family:     Seagate Barracuda 7200.14 (AF)
Device Model:     ST2000DM001-1CH164
Serial Number:    Z1E1DWN4
LU WWN Device Id: 5 000c50 04e53cf8d
Firmware Version: CC43
User Capacity:    2,000,398,934,016 bytes [2.00 TB]
Sector Sizes:     512 bytes logical, 4096 bytes physical
Rotation Rate:    7200 rpm
Form Factor:      3.5 inches
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ATA8-ACS T13/1699-D revision 4
SATA Version is:  SATA 3.0, 6.0 Gb/s (current: 3.0 Gb/s)
Local Time is:    Thu Nov 30 00:56:27 2017 UTC
SMART support is: Available - device has SMART capability.
SMART support is: Enabled

SMART Attributes Data Structure revision number: 10
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  1 Raw_Read_Error_Rate     0x000f   113   099   006    Pre-fail  Always       -       53347512
  3 Spin_Up_Time            0x0003   095   095   000    Pre-fail  Always       -       0
  4 Start_Stop_Count        0x0032   100   100   020    Old_age   Always       -       9
  5 Reallocated_Sector_Ct   0x0033   100   100   036    Pre-fail  Always       -       0
  7 Seek_Error_Rate         0x000f   083   060   030    Pre-fail  Always       -       4510892460
  9 Power_On_Hours          0x0032   087   087   000    Old_age   Always       -       12060
 10 Spin_Retry_Count        0x0013   100   100   097    Pre-fail  Always       -       0
 12 Power_Cycle_Count       0x0032   100   100   020    Old_age   Always       -       9
183 Runtime_Bad_Block       0x0032   100   100   000    Old_age   Always       -       0
184 End-to-End_Error        0x0032   100   100   099    Old_age   Always       -       0
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always       -       0
188 Command_Timeout         0x0032   100   100   000    Old_age   Always       -       0 0 0
189 High_Fly_Writes         0x003a   099   099   000    Old_age   Always       -       1
190 Airflow_Temperature_Cel 0x0022   074   065   045    Old_age   Always       -       26 (Min/Max 23/35)
191 G-Sense_Error_Rate      0x0032   100   100   000    Old_age   Always       -       0
192 Power-Off_Retract_Count 0x0032   100   100   000    Old_age   Always       -       9
193 Load_Cycle_Count        0x0032   100   100   000    Old_age   Always       -       269
194 Temperature_Celsius     0x0022   026   040   000    Old_age   Always       -       26 (0 19 0 0 0)
197 Current_Pending_Sector  0x0012   100   100   000    Old_age   Always       -       24
198 Offline_Uncorrectable   0x0010   100   100   000    Old_age   Offline      -       24
199 UDMA_CRC_Error_Count    0x003e   200   200   000    Old_age   Always       -       0
240 Head_Flying_Hours       0x0000   100   253   000    Old_age   Offline      -       12059h+46m+50.704s
241 Total_LBAs_Written      0x0000   100   253   000    Old_age   Offline      -       67220217075
242 Total_LBAs_Read         0x0000   100   253   000    Old_age   Offline      -       7368543577

SMART Error Log Version: 1
No Errors Logged
```

SATA SSDs usually have rather different SMART values since many of the regular values do not apply. Most SSDs will provide pairs of counters for the total amount of reads and writes that have been completed, allowing the administrator to track the wear lifetime of the device. Some SSDs even provide a drive lifetime statistic, as a %, either counting up or down toward the end of the useful life of the device. Sometimes the "raw value" has little meaning, and you need to look at the 'value' and 'thresh(old)' volumes instead. This SSD has relatively little wear:

```
#smartctl -a /dev/ada1
=== START OF INFORMATION SECTION ===
Model Family:     Intel 730 and DC S35x0/3610/3700 Series SSDs
Device Model:     INTEL SSDSC2BA200G4
Serial Number:    BTHV515103FW200MGN
LU WWN Device Id: 5 5cd2e4 04b7d7610
Firmware Version: G2010110
User Capacity:    200,049,647,616 bytes [200 GB]
Sector Sizes:     512 bytes logical, 4096 bytes physical
Rotation Rate:    Solid State Device
Form Factor:      2.5 inches
Device is:        In smartctl database [for details use: -P show]
ATA Version is:   ACS-2 T13/2015-D revision 3
SATA Version is:  SATA 2.6, 6.0 Gb/s (current: 6.0 Gb/s)
Local Time is:    Thu Nov 30 00:58:57 2017 UTC
SMART support is: Available - device has SMART capability.
SMART support is: Enabled

SMART Attributes Data Structure revision number: 1
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0032   100   100   000    Old_age   Always       -       0
  9 Power_On_Hours          0x0032   100   100   000    Old_age   Always       -       18834
 12 Power_Cycle_Count       0x0032   100   100   000    Old_age   Always       -       35
170 Available_Reservd_Space 0x0033   100   100   010    Pre-fail  Always       -       0
171 Program_Fail_Count      0x0032   100   100   000    Old_age   Always       -       0
172 Erase_Fail_Count        0x0032   100   100   000    Old_age   Always       -       0
174 Unsafe_Shutdown_Count   0x0032   100   100   000    Old_age   Always       -       20
175 Power_Loss_Cap_Test     0x0033   100   100   010    Pre-fail  Always       -       5290 (71 4859)
183 SATA_Downshift_Count    0x0032   100   100   000    Old_age   Always       -       0
184 End-to-End_Error        0x0033   100   100   090    Pre-fail  Always       -       0
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always       -       0
190 Temperature_Case        0x0022   064   060   000    Old_age   Always       -       36 (Min/Max 31/40)
192 Unsafe_Shutdown_Count   0x0032   100   100   000    Old_age   Always       -       20
194 Temperature_Internal    0x0022   100   100   000    Old_age   Always       -       36
197 Current_Pending_Sector  0x0012   100   100   000    Old_age   Always       -       0
199 CRC_Error_Count         0x003e   100   100   000    Old_age   Always       -       0
225 Host_Writes_32MiB       0x0032   100   100   000    Old_age   Always       -       655468
226 Workld_Media_Wear_Indic 0x0032   100   100   000    Old_age   Always       -       552
227 Workld_Host_Reads_Perc  0x0032   100   100   000    Old_age   Always       -       29
228 Workload_Minutes        0x0032   100   100   000    Old_age   Always       -       1129889
232 Available_Reservd_Space 0x0033   100   100   010    Pre-fail  Always       -       0
233 Media_Wearout_Indicator 0x0032   100   100   000    Old_age   Always       -       0
234 Thermal_Throttle        0x0032   100   100   000    Old_age   Always       -       0/0
241 Host_Writes_32MiB       0x0032   100   100   000    Old_age   Always       -       655468
242 Host_Reads_32MiB        0x0032   100   100   000    Old_age   Always       -       275081

SMART Error Log Version: 1
No Errors Logged
```

INTERACTIVE PERFORMANCE MONITORING

Now that it is established that the pool is healthy, it is time to look at what is happening with the pool. These interactive monitoring tools shed light on the operations that are being performed in real time.

zpool iostat

The zpool iostat command will print data about the activity on the pool. It shows the number of read and write IOPS, as well as bytes per second. If no other parameters are given, it will display one status line for each pool. If a pool name is given, it will show only that pool. If an integer is given, it will run continuously, and print new stats every X seconds, where X is that integer. You'll notice the natural cycle of ZFS, where there are a minimal number of synchronous writes as requested by applications; then every 5 seconds all other buffered asynchronous writes are flushed out to disk. If you change the integer to a longer interval, it will provide an average over that time span.

```
# zpool iostat sestore5 1
               capacity     operations     bandwidth
pool         alloc   free   read  write   read  write
----------   -----  -----  -----  -----  -----  -----
sestore5     46.0T  84.5T     99    189  14.8M  4.89M
sestore5     46.0T  84.5T      9    208  9.81M  1.09M
sestore5     46.0T  84.5T      0      0  31.9K      0
sestore5     46.0T  84.5T    103      0  12.9M      0
sestore5     46.0T  84.5T     64      0  7.79M      0
sestore5     46.0T  84.5T     38    570  4.74M  16.4M
sestore5     46.0T  84.5T     34    152  5.23M   826K
sestore5     46.0T  84.5T     11      0   278K      0
sestore5     46.0T  84.5T      6      0   247K      0
sestore5     46.0T  84.5T    146      0  16.4M      0
sestore5     46.0T  84.5T     31    977  1.46M  8.85M
sestore5     46.0T  84.5T      3      0   487K  3.99K
sestore5     46.0T  84.5T     35      0  4.12M      0
sestore5     46.0T  84.5T      1      0  2.00M      0
sestore5     46.0T  84.5T     12      0  63.9K      0
sestore5     46.0T  84.5T    244    650   978K  8.25M
sestore5     46.0T  84.5T      5      0   235K      0
sestore5     46.0T  84.5T      0      0      0      0


# zpool iostat sestore5 30
                 capacity     operations     bandwidth
pool         alloc   free   read  write   read  write
----------    -----  -----  -----  -----  -----  -----
sestore5     46.0T  84.5T     95    179  14.0M  4.93M
sestore5     46.0T  84.5T     22    163  5.43M  4.76M
sestore5     46.0T  84.5T      3    161  2.04M  5.96M
sestore5     46.0T  84.5T     17    264  3.71M  3.96M
sestore5     46.0T  84.5T     13    279  3.54M  4.01M
sestore5     46.0T  84.5T     21    416  4.87M  4.80M
sestore5     46.0T  84.5T     19    152  4.62M  6.89M
sestore5     46.0T  84.5T     36    166  6.21M  5.26M
sestore5     46.0T  84.5T     15    125  4.19M  2.66M
sestore5     46.0T  84.5T     16    136  3.54M  4.54M
```

top -m io

One of the fastest ways to figure out which application is causing all of the I/O is to use top. On FreeBSD top has a -m flag to change the mode. In IO mode, instead of tracking applications by CPU and memory usage, it tracks reads, writes, and other IO operations. This can help you determine which application is consuming all of the IO resources. To break things down further, see the section on the DTrace Toolkit.

```
# top -m io -o read
…
  PID USERNAME        VCSW  IVCSW   READ  WRITE  FAULT  TOTAL PERCENT COMMAND
35765 www              931     85     48      0      0     48  10.50% nginx
35766 www              410     66     32      0      0     32   7.00% nginx
 4994 root           66139   3971     10      0      0     10   2.19% nfsd
35248 www             1113     39      6    143      0    149  32.60% nginx
 3975 mysql            205     28      1    145      0    146  31.95% mysqld
```

zfs-stats

On Solaris, ZFS uses a system called kstat to publish various statistics about what is happening internally in ZFS. On FreeBSD those stats are published via the sysctl interface. The **`sysutils/zfs-stats`** package can summarize those statistics in a more human-readable way, logically grouping them together.

```
# zfs-stats -A
------------------------------------------------------------------------
ZFS Subsystem Report                            Thu Oct 19 03:50:59 2017
------------------------------------------------------------------------

ARC Summary: (HEALTHY)
        Memory Throttle Count:                  0

ARC Misc:
        Deleted:                                92.50m
        Recycle Misses:                         0
        Mutex Misses:                           14.31k
        Evict Skips:                            6.79k

ARC Size:                               99.87%  31.92   GiB
        Target Size: (Adaptive)         100.00% 31.96   GiB
        Min Size (Hard Limit):          12.50%  4.00    GiB
        Max Size (High Water):          8:1     31.96   GiB

ARC Size Breakdown:
        Recently Used Cache Size:       98.18%  31.38   GiB
        Frequently Used Cache Size:     1.82%   594.11  MiB

ARC Hash Breakdown:
        Elements Max:                           690.77k
        Elements Current:               75.55%  521.90k
        Collisions:                             5.89m
        Chain Max:                              4
        Chains:                                 8.13k
```

MEMORY THROTTLE

The more important indicator of problems is the "Memory Throttle Count." This is the number of times that the ZFS ARC has had to reduce its memory usage because of demands elsewhere in the system. You might consider setting the maximum size of the ARC **(vfs.zfs.arc_max)** to a value that makes ZFS coexist with your other workloads better. The output also shows a breakdown of the ARC usage by MRU (Most Recently Used) and MFU (Most Frequently Used). This gives you some insight into how the cache is adapting to the workload.

zfs-mon

The **sysutils/zfs-stats** package also includes a second tool, zfs-mon, which looks at how a subset of the kstats are changing over time. This can provide useful insight into how the requests are being broken down, and how the various caching layers in ZFS are being used. The stats break down the performance of the ARC, L2ARC, the filesystem prefetch, and the device prefetching code. It also breaks down data vs metadata operations. By default, ZFS limits the amount of cache available for metadata to 25% of the max ARC size. If the total storage capacity is very large—and most operations impact only the metadata of the files, not the content—increasing the amount of the ARC that can be used for metadata can actually increase performance, since otherwise the ARC may be 3/4s full of content that will not be referenced again before it is replaced with other content.

```
# zfs-mon -a
ZFS real-time cache activity monitor
Seconds elapsed: 329

Cache hits and misses:
                                    1s    10s    60s    tot
                      ARC hits:    130    305    591   2875
                    ARC misses:     38    113     62    142
          ARC demand data hits:     95    236    539   2642
        ARC demand data misses:      2     51     29     18
      ARC demand metadata hits:     34     46     36    207
    ARC demand metadata misses:      0     36     17     90
        ARC prefetch data hits:      1     17     13     23
      ARC prefetch data misses:     36     26     16     33
    ARC prefetch metadata hits:      0      6      3      2
  ARC prefetch metadata misses:      0      1      0      0
                   ZFETCH hits:     28     69     50    189
                 ZFETCH misses:  18639  17450  18029  23507
           VDEV prefetch hits:       0      3      1      5
         VDEV prefetch misses:       0     39     12     18

Cache efficiency percentage:
                           10s    60s    tot
                  ARC:   72.97  90.51  95.29
      ARC demand data:   82.23  94.89  99.32
  ARC demand metadata:   56.10  67.92  69.70
    ARC prefetch data:   39.53  44.83  41.07
ARC prefetch metadata:   85.71 100.00 100.00
               ZFETCH:    0.39   0.28   0.80
        VDEV prefetch:    7.14   7.69  21.74
```

As you can see, the ARC cache hit ratio varies quite a lot over short intervals, but in the 5-1/2 minutes this tool ran, the overall average was a 95.29% hit ratio.

GEOM STATS

gstat is an interactive tool that pulls statistics from the FreeBSD GEOM subsystem. It can be a useful window into what is happening with the underlying storage. For each GEOM object (there may be many that represent a single device, or a partition or other subdivision of a device), the depth of the queue, total operations per second, read operations per second, read kilobytes per second, milliseconds per read operation, and all the same again for write operations are printed. Then a synthesized '% busy' number is calculated, a best guess only, and can often be seen exceeding 100%. There are additional operation types (delete for TRIM/UNMAP etc., and flush) that can be shown with additional flags. If the sum of the read and write IOPS per second is less than the value in the ops/s column, it is likely that these other operations are happening as well.

```
# gstat -f da..\?$
 L(q)  ops/s    r/s    kBps   ms/r    w/s    kBps   ms/w   %busy Name
  0       0      0      0    0.0      0       0    0.0     0.0| ada0
  0       0      0      0    0.0      0       0    0.0     0.0| ada1
  5     733      0      0    0.0    733   62244   10.5    88.8| ada2
  7     883      0      0    0.0    883   62443    7.1    85.9| ada3
  7     961      0      0    0.0    961   62539    5.2    61.6| ada4
  0     960      0      0    0.0    960   63425    8.6    73.4| ada5
  7    1047      0      0    0.0   1047   65006    5.5    79.1| da0
 10    1078      0      0    0.0   1078   60751    5.9    81.4| da1
```

DTRACE TOOLKIT

DTrace is a very powerful tool designed to allow you to safely inspect and debug the running system, while having very minimal impact on performance when not debugging. DTrace scripts vary in complexity, from simple one liners to interactive tools.

A simple example of a DTrace one liner:

```
# dtrace -n 'syscall::read:entry { @bytes[execname] = sum(arg2); }'
```

This creates an aggregation of the 3rd argument (they are numbered from 0) of the read system call, by the calling application's name. Run this for a few seconds, then hit control+c to stop it. It will then print out a list of every application that called read, and the total number of bytes that were read. Now it is obvious which application was causing all of the reads from disk.

You don't have to write your own DTrace scripts; the OpenDTrace project maintains a collection of cross-platform scripts that you can download and use. These serve as a great starting point that can be modified to answer the questions you want to ask; https://github.com/opendtrace/toolkit

To look at how much data is being written in each transaction group, or how long each transaction group is taking to sync to disk, check out these DTrace examples by Adam Leventhal: http://dtrace.org/blogs/ahl/2014/08/31/openzfs-tuning/

CONTINUOUS PERFORMANCE MONITORING

Understanding the cause of performance problems first requires having something to compare the new measurements and observations against. Is the current level of operations per second typical? Or is it much higher or lower than expected. In order to make sense of the cache hit ratio, you need to know what it is when the system is NOT having problems. In order to have this information, and to be able to make sense of it, you need to be continuously recording the metrics that you will want to compare the current state of the system against. Collecting, storing, and graphing these metrics in a useful way is the key to being able to quickly diagnose issues and detect problems early.

Disks can be very ungentlemanly when they fail. Rather than loudly dying and going offline completely, they often misbehave. One of the first signs that a disk is beginning to fail can be greatly increased read and write latencies. Consumer grade disks often retry internally many times before returning a read error. The operating system might then helpfully ask the drive to retry a few more times, each of those repeated commands resulting in a series of additional internal retries. Because of this, operating systems will often have rather high timeouts while waiting for commands to complete, with defaults on the order of 30 seconds per command and 5 retries. A single failed read can thus hold up the entire system for 2-1/2 minutes.

ZFS KSTATS

ZFS presents an impressive number of stats and counters via the kstat interface. On FreeBSD, this is currently exposed via the kstats.zfs sysctl mibs.

One of the advantages of ZFS is the ARC (Adaptive Replacement Cache), which provides better cache hit ratios than a standard LRU (Least Recently Used) cache. Looking at the various stats about the ARC can provide insight into what is happening with a system.

- `kstat.zfs.misc.arcstats.c_max` — The target maximum size of the ARC.
- `kstat.zfs.misc.arcstats.c_min` — The target minimum size of the ARC. The ARC will not shrink below this size, although it can be adjusted with the `vfs.zfs.arc_min` sysctl.
- `kstat.zfs.misc.arcstats.size` — The current size of the ARC; if this is less than the maximum, your system has either not had enough activity to fill the ARC, or memory pressure from other processes has caused the ARC to shrink.
- `kstat.zfs.misc.arcstats.c` — The current target size of the ARC. If the current size of the ARC is less than this value, the ARC will try to grow.
- `kstat.zfs.misc.arcstats.p` — How much of the ARC is to be used for the MRU list; the remainder is the target for the MFU list. This value will adjust dynamically based on workload. A lower value suggests frequent access to the same blocks, where a higher value suggests a more varied workload.
- `kstat.zfs.misc.arcstats.arc_meta_used` — The amount of the ARC used to store metadata rather than user data. If this value has reached `vfs.zfs.arc_meta_limit` (which defaults to 25% of `vfs.zfs.arc_max`), then consider raising or lowering the fraction of the ARC used for metadata. Caching more metadata will increase the speed of directory scans and other operations, at the cost of decreasing the amount of user data that can be cached.

SNMP

net-snmpd provides a number of useful counters like total IOPS per device and bytes read and written. These can be used to create graphs to provide some historical perspective when looking for performance problems. Is the IOPS load twice what it normally is? That might be your problem. If the number of IOPS is down, but the workload is higher, something might be causing one or more devices to perform suboptimally.

OTHER TOOLS

There are many different solutions for monitoring, measuring, and recording statistics from a system. Some you might wish to investigate include:

- Zabbix — An advanced monitoring suite with some predefined probes for ZFS.
- Collectd — A metrics collection daemon that can be used with a number of different backends.
- Grafana — A graphing and analytics tool for time series data that can make sense of metrics gathered by applications such as collectd.
- OSQuery — An operating system instrumentation framework for analyzing the live and historical metrics of a system using a familiar structured query language. ●

ALLAN JUDE is VP of operations at ScaleEngine Inc., a global HTTP and Video Streaming Content Distribution Network, where he makes extensive use of ZFS on FreeBSD. Allan is a FreeBSD src and doc committer, and was elected to the FreeBSD core team in summer 2016. He is also the host of the weekly video podcast BSDNow.tv (with Benedict Reuschling), and coauthor of *FreeBSD Mastery: ZFS* and *FreeBSD Mastery: Advanced ZFS* with Michael W Lucas.