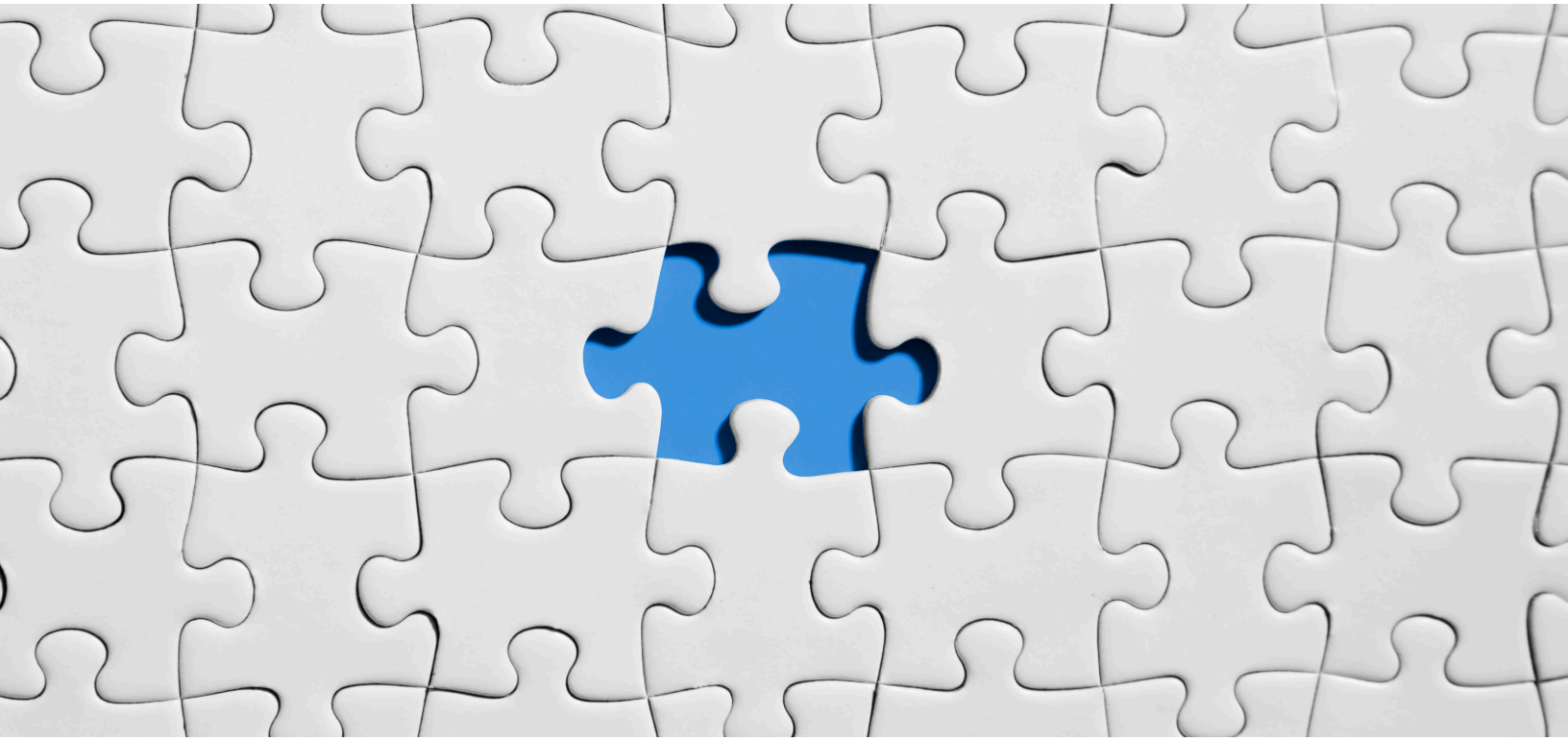
DELL Technologies

VALIDATION OF ISILON F800 FOR EDA WORKLOAD



Xiao-Jing Carroll Zhu

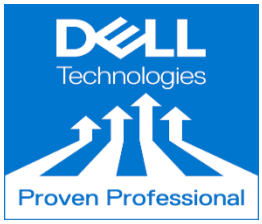
Senior System Engineer

Dell EMC

Xiaojing.zhu@dell.com

Knowledge Sharing Article

DELL Technologies Proven Professional



The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged/Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage
- Enterprise Architect

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

Learn more at www.dell.com/certification

Table of Contents

Disclaimer: The views, processes or methodologies published in this article are those of the author. They do not necessarily reflect Dell Technologies' views, processes or methodologies.

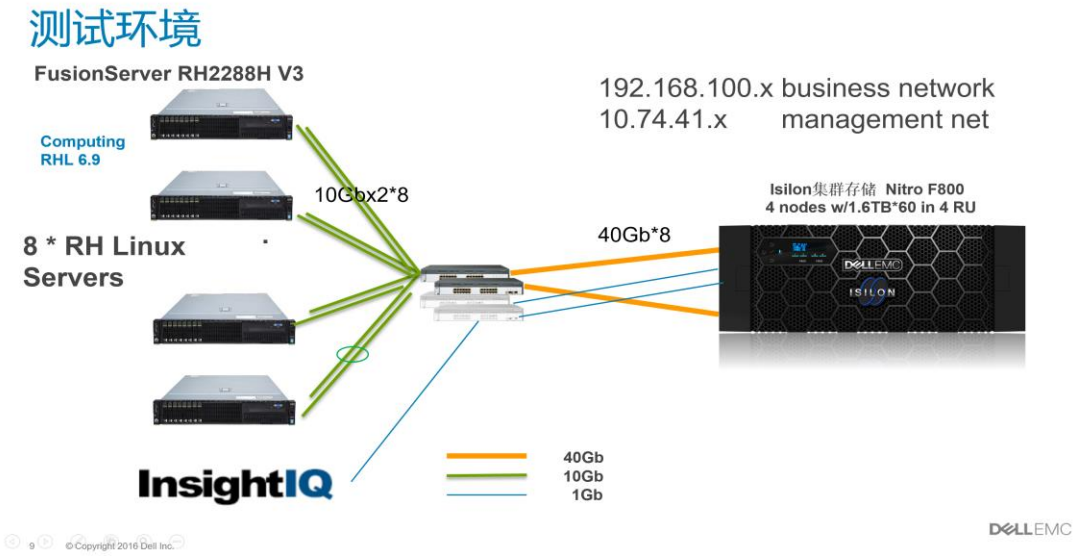
F800 System Configuration

Evaluation Configuration	Qty & Description
Node Models	**Isilon F800 All-flash array** 4xF800 nodes, each with: 1x16 core Intel Broadwell EP Xeon E5-2697A v4@ 2.6 GHz CPU 256 GB RAM 2x40 Gbps QDR Infiniband network ports (Internal back-end network) 2x40 Gbps Ethernet network ports (Front-end client network) 15x1.6 TB SSD
IB Switch Model	2x 36-port Infiniband Switches
Software Modules (License Keys)	SmartConnect Advanced,SmartQuotas,SnapshotIQ,SyncIQ,InsightIQ
OneFS Version	OneFS 8.1.2.0
SSD Usage	Storage
Client Protocols	NFSv3
Network	40GbE

"Clients" Systems Configuration

Evaluation Configuration	Qty & Description
Huawei FusionServer RH2288H V3	CPU Xeon E5-2697A v4@2.6GHz MEM 32GB*24 HDD SAS600GB*8 RAID 5, 1 HOSSPARE
Benchmark Software	vdbench v5.04.06

Network Diagram of the Testing Environment



Client servers list: (Each server configured with two 10GbE NICs and two subnet IP address)

Client 1# szhpctest01	RHEL 6.9，192.168.100.200 192.168.200.200
Client 2# szhpctest02	RHEL 6.9，192.168.100. 201 192.168.200.201
Client 3# szhpctest03	RHEL 6.9，192.168.100. 202 192.168.200.202
Client 4# szhpctest04	RHEL 6.9，192.168.100. 203 192.168.200.203
Client 5# szhpctest05	RHEL 6.9，192.168.100. 204 192.168.200.204
Client 6# szhpctest06	RHEL 6.9，192.168.100. 205 192.168.200.205
Client 7# szhpctest07	RHEL 6.9，192.168.100. 206 192.168.200.206
Client 8# szhpctest08	RHEL 6.9，192.168.100. 207 192.168.200.207

Client server's configurations

```
[root@szhpctest01 vdbench]#
[root@szhpctest01 vdbench]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.74.43.200 szhpctest01
10.74.43.201 szhpctest02
10.74.43.202 szhpctest03
10.74.43.203 szhpctest04
10.74.43.204 szhpctest05
10.74.43.205 szhpctest06
10.74.43.206 szhpctest07
10.74.43.207 szhpctest08
[root@szhpctest01 vdbench]# ifconfig -a | grep 200
          Memory:92200000-922fffff
          inet addr:10.74.43.200  Bcast:10.74.43.223  Mask:255.255.255.224
          inet addr:192.168.100.200  Bcast:192.168.100.255  Mask:255.255.255.0
          inet addr:192.168.200.200  Bcast:192.168.200.255  Mask:255.255.255.0
[root@szhpctest01 vdbench]# df -h | grep hw
hisi200.isi:/ifs/hw200
                      85T   29T   54T  35% /hw200
hisi100.isi:/ifs/hw100
                      85T   29T   54T  35% /hw100
[root@szhpctest01 vdbench]# cat /proc/m
mdstat   meminfo  misc     modules  mounts   mtd      mtrr
[root@szhpctest01 vdbench]# cat /proc/meminfo
MemTotal:       793616352 kB
```

F800's cluster status

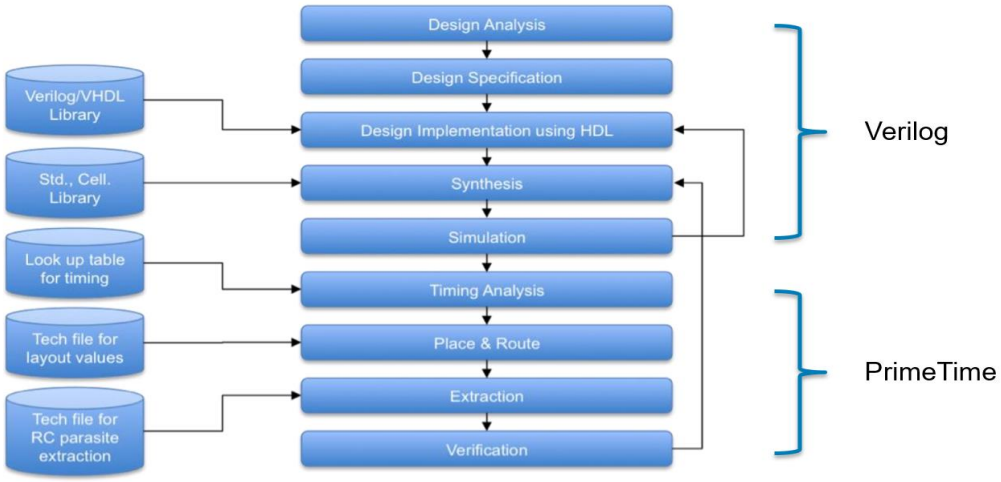
```
HUAWEI-1# isi status -q
Cluster Name: HUAWEI
Cluster Health:     [ ATTN]
Cluster Storage:  HDD                 SSD Storage
Size:             0 (0 Raw)           81.6T (84.7T Raw)
VHS Size:         3.1T
Used:             0 (n/a)             28.4T (35%)
Avail:            0 (n/a)             53.2T (65%)

                   Health  Throughput (bps)  HDD Storage      SSD Storage
ID |IP Address     |DASR |  In   Out  Total| Used / Size     |Used / Size
---+---------------+-----+-----+-----+-----+-----------------+-----------------
  1|10.74.41.150   |-A-- |    0| 4.4M| 4.4M|(No Storage HDDs)| 7.1T/20.8T( 34%)
  2|10.74.41.151   | OK  |    0|    0|    0|(No Storage HDDs)| 7.1T/19.4T( 37%)
  3|10.74.41.197   | OK  |    0|74.6k|74.6k|(No Storage HDDs)| 7.1T/20.8T( 34%)
  4|10.74.41.198   | OK  |    0|    0|    0|(No Storage HDDs)| 7.1T/20.8T( 34%)
---+---------------+-----+-----+-----+-----+-----------------+-----------------
Cluster Totals:          |    0| 4.5M| 4.5M|    0/    0( n/a)|28.4T/81.6T( 35%)

     Health Fields: D = Down, A = Attention, S = Smartfailed, R = Read-Only
HUAWEI-1#
HUAWEI-1# isi version
Isilon OneFS v8.1.2.0 B_8_1_2_016(RELEASE): 0x8010250000000010:Wed Aug  1 16:36:13 PDT 2018
HUAWEI-1#
HUAWEI-1#
HUAWEI-1# isi network interfaces list
LNN  Name     Status  Owners                  IP Addresses
-------------------------------------------------------------
1    40gige-1 Up      groupnet0.subnet0.pool0 192.168.100.11
1    40gige-2 Up      groupnet0.subnet0.pool1 192.168.200.11
1    mgmt-1   Up      groupnet0.subnet1.pool0 10.74.41.150
2    40gige-1 Up      groupnet0.subnet0.pool0 192.168.100.12
2    40gige-2 Up      groupnet0.subnet0.pool1 192.168.200.12
2    mgmt-1   Up      groupnet0.subnet1.pool0 10.74.41.151
3    40gige-1 Up      groupnet0.subnet0.pool0 192.168.100.13
3    40gige-2 Up      groupnet0.subnet0.pool1 192.168.200.13
3    mgmt-1   Up      groupnet0.subnet1.pool0 10.74.41.197
4    40gige-1 Up      groupnet0.subnet0.pool0 192.168.100.14
4    40gige-2 Up      groupnet0.subnet0.pool1 192.168.200.14
4    mgmt-1   Up      groupnet0.subnet1.pool0 10.74.41.198
-------------------------------------------------------------
Total: 12
HUAWEI-1#
```

F800's node hardware configuration

```
Product: F800-4U-Single-256GB-1x1GE-2x40GE SFP+-24TB SSD
  HWGen: PSI (PSI Hardware)
Chassis: INFINITY (Infinity chassis)
    CPU: GenuineIntel (2.60GHz, stepping 0x000406f1)
   PROC: Single-proc, 16-HT-core
    RAM: 274700771328 Bytes
   Mobo: EMCInfinityMobo (Custom EMC Motherboard)
  NVRam: INFINITY (Infinity Memory Journal) (8192MB card) (size 8589934592B)
 DskCtl: PMC8074 (PMC 8074) (36 ports)
 DskExp: PMC8056I (PMC-Sierra PM8056 - Infinity)
```

EDA TEST

EDA Workflow



EDA design is storage I/O-intensive. As designs move to smaller physical silicon geometries, design complexity and storage requirements for the corresponding EDA tool flow grows exponentially. As the EDA tools are often accessing storage shared by other tools within the EDA flow, all tools can be impacted by any tool that induces an I/O bottleneck. Front-end tools are particularly sensitive to massively parallel computing (MPP).

Front-end (Design Verification) tool flow

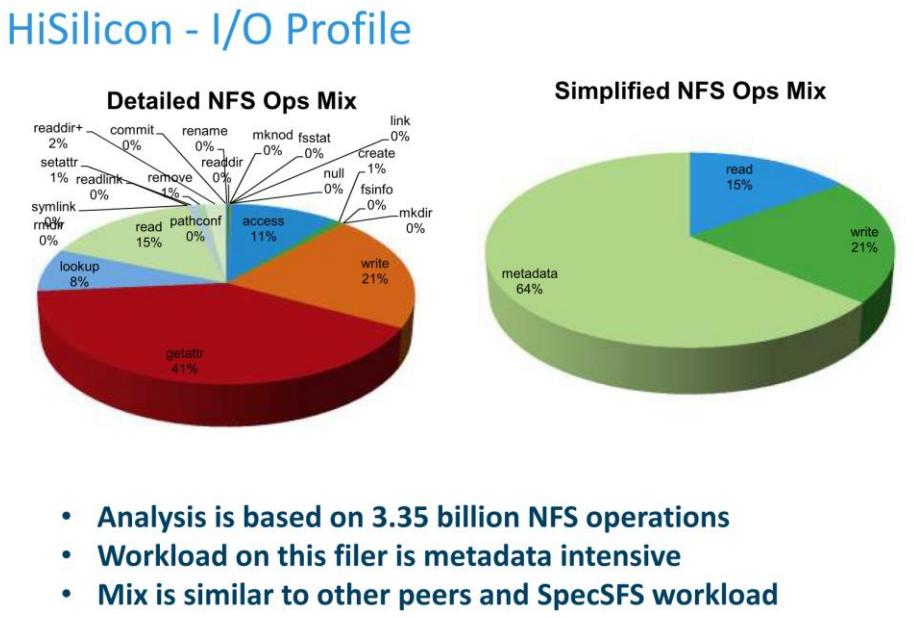
Design simulation is typically a batch process wherein 1000's of cores are used to batch process 10's of thousands of simulation jobs. These jobs are high performance, high concurrency computing jobs and require very fast storage that can handle very high I/O loads at scale. Other tools with intensive compute requirements include emulation compilation, analog/mixed-signal simulation. In Hisilicon, it is Verilog.

Back-end tools (Design Implementation and physical verification)

These tools are characterized by increasingly larger individual files (but fewer of them). Tools include design synthesis, timing analysis, floor planning, place and route, design rule and electrical rule checks, and equivalency checking. These tools tend to generate large files that are sequential in nature. The data, however, is often located on storage shared with front-end tools, so storage must also support high IOPs without bottlenecks. It is PrimeTime in Hisilicon.

HiSilicon Workload Analysis

According to Jig's file (HiSilicon workload analyst.pdf, see https://isilon.my.salesforce.com/00P39000016G1AW), HiSilicon's workload is metadata- intensive, so the test should simulate it.



- **Analysis is based on 3.35 billion NFS operations**
- **Workload on this filer is metadata intensive**
- **Mix is similar to other peers and SpecSFS workload**

We recommended the benchmark method to the customer. They were previously using SIO from NetApp. It's bad to simulate EDA workloads since it is just a general purpose I/O load generator. It performs synchronous I/O's to the specified file(s). Finally, we persuaded the customer about realistic benchmarking and understanding their workflow.

Tuning Preparation

- Vdbench profiles defined in the directory /mnt/ision/test_profiles
  - hosts_6.txt - hosts_6 defines 6 JVMs per host
  - nitro_fsd.txt – defines the directory structure.
  - Profiles.txt – defines all the workloads and running jobs.

  # Hosts definition

  # Default parameters

  hd=default,vdbench=/root/vdbench,shell=vdbench

  hd=client1_1,system=192.168.100.200

hd=client1_2,system=192.168.100.200

hd=client1_3,system=192.168.100.200

hd=client1_4,system=192.168.100.200

hd=client1_5,system=192.168.100.200

hd=client1_6,system=192.168.100.200

hd=client2_1,system=192.168.100.201

hd=client2_2,system=192.168.100.201

hd=client2_3,system=192.168.100.201

hd=client2_4,system=192.168.100.201

hd=client2_5,system=192.168.100.201

hd=client2_6,system=192.168.100.201

hd=client3_1,system=192.168.100.202

hd=client3_2,system=192.168.100.202

hd=client3_3,system=192.168.100.202

hd=client3_4,system=192.168.100.202

hd=client3_5,system=192.168.100.202

hd=client3_6,system=192.168.100.202

hd=client4_1,system=192.168.100.203

hd=client4_2,system=192.168.100.203

hd=client4_3,system=192.168.100.203

hd=client4_4,system=192.168.100.203

hd=client4_5,system=192.168.100.203

hd=client4_6,system=192.168.100.203

hd=client5_1,system=192.168.100.204

hd=client5_2,system=192.168.100.204

hd=client5_3,system=192.168.100.204

hd=client5_4,system=192.168.100.204

hd=client5_5,system=192.168.100.204

hd=client5_6,system=192.168.100.204

hd=client6_1,system=192.168.100.205

hd=client6_2,system=192.168.100.205

hd=client6_3,system=192.168.100.205

hd=client6_4,system=192.168.100.205

hd=client6_5,system=192.168.100.205

hd=client6_6,system=192.168.100.205

hd=client7_1,system=192.168.100.206

hd=client7_2,system=192.168.100.206

hd=client7_3,system=192.168.100.206

hd=client7_4,system=192.168.100.206

hd=client7_5,system=192.168.100.206

hd=client7_6,system=192.168.100.206

hd=client8_1,system=192.168.100.207

hd=client8_2,system=192.168.100.207

hd=client8_3,system=192.168.100.207

hd=client8_4,system=192.168.100.207

hd=client8_5,system=192.168.100.207

hd=client8_6,system=192.168.100.207

Below is the NITRO_FSD.TXT

Verilog/software like file system. High file counts with small files

#

distribution=all,width=4,depth=8,files=1500 will create (((4^(8 + 1)) - 1 / (4 - 1)) - 1) * 1200 = 104,856,000 files @ 25.6 Kib avg = 2560 GiB

width=4,depth=7,files=20 = 4 ^ 7 * 20 = 327680 files @ 25.6 KiB avg = 8 GiB

#

The following FSD uses the $host variable which will cause the value to be replaced by each host name. Be careful with the number of hosts used this way. If you have 8 hosts and 6 JVM per host for 48 process, then the below setting will create 327680 files/process * 48 process = 15728640 total files / fsd, there are 2 fsd will get total files = 31457280

fsd=fs_eda_verilog_1_$host,anchor=/hw100/isilon/test/verilog/$host,width=4,depth=7,files=20,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

fsd=fs_eda_verilog_2_$host,anchor=/hw200/isilon/test/verilog/$host,width=4,depth=7,files=20,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

Shared file system versions

#
fsd=fs_eda_verilog_1,anchor=/hw100/isilon/test/verilog,shared=yes,width=4,depth=8,files=20,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

#
fsd=fs_eda_verilog_2,anchor=/hw200/isilon/test/verilog,shared=yes,width=4,depth=8,files=20,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

#
fsd=fs_eda_verilog_1,anchor=/hw100/isilon/test/verilog,shared=yes,distribution=all,width=4,depth=8,files=1200,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

#
fsd=fs_eda_verilog_2,anchor=/hw200/isilon/test/verilog,shared=yes,distribution=all,width=4,depth=8,files=1200,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

Below is the Verilog small file system profile. It has fewer files but combined with other workloads, it is still very good.

distribution=all,width=4,depth=7,files=1500 will create (((4^(6 + 1)) - 1 / (4 - 1)) - 1) * 1500 = 32,766,000 files @ 25.6 Kib avg = 799 GiB

fsd=fs_eda_verilogsmall_1,anchor=/hw100/isilon/test/verilog_small,shared=yes,distribution=all,width=4,depth=7,files=1500,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

fsd=fs_eda_verilogsmall_2,anchor=/hw200/isilon/test/verilog_small,shared=yes,distribution=all,width=4,depth=7,files=1500,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)

Primetime like file system (large files)

2^8 = 256 * 10 files = 2560 files @ 3 GiB/file = 7680 GB

6^2*2 = 64 * 10 files*2 = 1280 files @ 3 GiB/file = 7680 GB

fsd=fs_eda_pt_1,anchor=/hw100/isilon/test/pt,shared=yes,width=8,depth=2,files=10,size=(1G,50,5G,50)

fsd=fs_eda_pt_2,anchor=/hw200/isilon/test/pt,shared=yes,width=8,depth=2,files=10,size=(1G,50,5G,50)

Primetime like file system (100GB size large files)

2^4 = 16*6 files @ 95 GiB/file = 9120 GB

fsd=fs_eda_pt_large_1,anchor=/hw100/isilon/test/pt_large,shared=yes,width=2,depth=4,files=6,size=(90G,50,100G,50)

fsd=fs_eda_pt_large_2,anchor=/hw200/isilon/test/pt_large,shared=yes,width=2,depth=4,files=6,size=(90G,50,100G,50)

General medium-sized files for throughput and filling up storage.

width=10,depth=4,files= will create 10^4 * 10 files = 1,000,000 files @ 100 MiB each is approx 9.5 TiB

fsd=fs_eda_io_1,anchor=/hw100/isilon/test/throughput,shared=yes,width=10,depth=4,files=10,size=100M

fsd=fs_eda_io_2,anchor=/hw200/isilon/test/throughput,shared=yes,width=10,depth=4,files=10,size=100M

Mount a file system of Isilon on each client by 2:1 mapping of host to every node

Below is the profile.txt

```
# ====================
# General settings
# ====================
Setting messages can=no. This should only be done after testing shows you can ignore /var/log/messages output from the clients. Sometimes the output is useful for identifying issues with the benchmark setup messages can=no


# ====================
# Host definition
#
This section imports a list of hosts. The number of hosts is very important as it defines how many machines will be running the benchmarks
as well as for some workflows, It will determine the number of file systems that are defined. Be careful to look for FSD (file system definitions) that use the $host parameters
# ====================
# hosts_4 defines 4 JVMs per host
# include=/mnt/isilon/test_profiles/hosts_4.txt
# hosts_6 defines 6 JVMs per host
include=/root/vdbench/hosts_4.txt
#include=/root/vdbench/hosts_6.txt
# ====================
# File system definition
#
When defining the directory structure, be careful with the width and depth numbers. By default, only files at the leaf of the directory structure are populated with files To calculate the number of leaf directories the formula is: width ^ depth e.g. width=4 and depth=3, 4^3 = 64. Each directory will have a number of files as configured, e.g. width=4,depth=3,files=100 would give a total of 64 * 100 files = 6,400 files in total. The total number of directories is greater as the calculation above only finds leaf directories. To find the total number of directories, minus the root, the calculation is:
( (width ^ (depth + 1)) - 1 / (width - 1) ) - 1
# ====================
include=/root/vdbench/nitro_fsd.txt

# ====================
# Workload definition
# ====================
#
Note: If you need to add file create/delete pairs, you can use the following options
     operation=write, fileio=(seq,delete).
```

Be aware that whatever skew you use, the number of ops you request will first be assigned to the main operation, in this case writes. The number of creates and deletes generated can vary depending on file size. For example, a OPS rate of 10,000 can result in the mix below. Test to see what type of skew setting you need when adding creates/deletes.

```
#       12,000 access operations
#       25,000 lookup operations
#       8600 remove operations
#       8600 create operations
#       8600 open operations
#       8600 close operations
#       8600 write operations
```

Configure the default workload settings here. A lot of I/O for systems is actually sequential in nature. What is random is the choice of files used.

```
fwd=default,fileio=sequential,fileselect=random,xfersizes=1M
```

Modify the defaults used by vdbench when it formats a file system. You must specify threads and the xfersize here. Only these 2 parameters make any difference to the formatting.

```
fwd=format,threads=32,xfersize=1024K
```

```
# =====
# EDA mix
# =====
fwd=eda_mix_1,fsd=(fs_eda_verilogsmall_1,fs_eda_io_1,fs_eda_pt_1),xfersize=64K,operation=getattr,skew=42
fwd=eda_mix_2,fsd=(fs_eda_verilogsmall_1,fs_eda_io_1,fs_eda_pt_1),xfersize=64K,operation=access,skew=19
fwd=eda_mix_3,fsd=(fs_eda_verilogsmall_1,fs_eda_io_1,fs_eda_pt_1),xfersize=64K,operation=setattr,skew=2
fwd=eda_mix_4,fsd=(fs_eda_verilogsmall_1,fs_eda_io_1,fs_eda_pt_1),xfersize=64K,operation=read,skew=16
fwd=eda_mix_5,fsd=(fs_eda_verilogsmall_1,fs_eda_io_1,fs_eda_pt_1),xfersize=64K,operation=write,skew=20
fwd=eda_mix_6,fsd=(fs_eda_verilogsmall_1,fs_eda_io_1,fs_eda_pt_1),xfersize=64K,operation=write,skew=1,fileio=(seq,delete)
fwd=eda_mix_7,fsd=(fs_eda_verilogsmall_2,fs_eda_io_2,fs_eda_pt_2),xfersize=64K,operation=getattr,skew=42
fwd=eda_mix_8,fsd=(fs_eda_verilogsmall_2,fs_eda_io_2,fs_eda_pt_2),xfersize=64K,operation=access,skew=19
fwd=eda_mix_9,fsd=(fs_eda_verilogsmall_2,fs_eda_io_2,fs_eda_pt_2),xfersize=64K,operation=setattr,skew=2
fwd=eda_mix_10,fsd=(fs_eda_verilogsmall_2,fs_eda_io_2,fs_eda_pt_2),xfersize=64K,operation=read,skew=16
fwd=eda_mix_11,fsd=(fs_eda_verilogsmall_2,fs_eda_io_2,fs_eda_pt_2),xfersize=64K,operation=write,skew=20
fwd=eda_mix_12,fsd=(fs_eda_verilogsmall_2,fs_eda_io_2,fs_eda_pt_2),xfersize=64K,operation=write,skew=1,fileio=(seq,delete)
```

```
# =====
# Verilog SMALL
# =====
fwd=eda_verilogsmall_1,fsd=fs_eda_verilogsmall_1,xfersize=64K,operation=getattr,skew=42
fwd=eda_verilogsmall_2,fsd=fs_eda_verilogsmall_1,xfersize=64K,operation=setattr,skew=18
fwd=eda_verilogsmall_3,fsd=fs_eda_verilogsmall_1,xfersize=64K,operation=read,skew=30
fwd=eda_verilogsmall_4,fsd=fs_eda_verilogsmall_1,xfersize=64K,operation=write,skew=9
fwd=eda_verilogsmall_5,fsd=fs_eda_verilogsmall_1,xfersize=64K,operation=write,skew=1,fileio
=(seq,delete)
fwd=eda_verilogsmall_6,fsd=fs_eda_verilogsmall_2,xfersize=64K,operation=getattr,skew=42
fwd=eda_verilogsmall_7,fsd=fs_eda_verilogsmall_2,xfersize=64K,operation=setattr,skew=18
fwd=eda_verilogsmall_8,fsd=fs_eda_verilogsmall_2,xfersize=64K,operation=read,skew=30
fwd=eda_verilogsmall_9,fsd=fs_eda_verilogsmall_2,xfersize=64K,operation=write,skew=9
fwd=eda_verilogsmall_10,fsd=fs_eda_verilogsmall_2,xfersize=64K,operation=write,skew=1,filei
o=(seq,delete)

# =====
```

```
# Verilog
# =====
```

This workload does not use a shared file system. Each process will create its own directory structure based on the host. This is done by adding the host= parameter to the definition. This will restrict this fwd to a specific host. However, because of this change, we cannot easily use the skew parameter to create workload percentages. To emulate the same behavior we create multiple fwd with the same operation thus manually creating a correct workload # mix. This method is more coarse and you can only easily do 10% or 5% skews of the workload. To do 10% skews you need 10 fwd lines and to do 5% skews you need 20 fwd lines This will also require a decrease in the number of threads to a small number like 1 or 2 as each host will have #fwd*#hosts*threads running. This needs to be set at the run definition.

```
fwd=eda_verilog_1_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=geta
ttr
fwd=eda_verilog_2_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=geta
ttr
fwd=eda_verilog_3_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=geta
ttr
fwd=eda_verilog_4_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=setat
tr
fwd=eda_verilog_5_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=setat
tr
fwd=eda_verilog_6_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=read
fwd=eda_verilog_7_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=read
fwd=eda_verilog_8_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=read
fwd=eda_verilog_9_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=writ
e
fwd=eda_verilog_10_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=wri
te
```

Below is a shared workload version, similar to the verilog SMALL

```
#fwd=eda_verilog_1,,fsd=fs_eda_verilog,xfersize=64k,operation=getattr,skew=30
#fwd=eda_verilog_2,fsd=fs_eda_verilog,xfersize=64k,operation=access,skew=2
#fwd=eda_verilog_3,fsd=fs_eda_verilog,xfersize=64k,operation=setattr,skew=18
#fwd=eda_verilog_4,fsd=fs_eda_verilog,xfersize=64k,operation=read,skew=30
#fwd=eda_verilog_5,fsd=fs_eda_verilog,xfersize=64k,operation=write,skew=20



# =====
# Primetime 96% write
# =====
#fwd=eda_pt_write_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=getattr,skew=1
#fwd=eda_pt_write_2,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=1
#fwd=eda_pt_write_3,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=write,skew=48
#fwd=eda_pt_write_4,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=getattr,skew=1
#fwd=eda_pt_write_5,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=1
#fwd=eda_pt_write_6,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=write,skew=48

fwd=eda_pt_write_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=write,skew=50
fwd=eda_pt_write_2,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=write,skew=50

# =====
# Primetime 98% read
# =====
#fwd=eda_pt_read_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=getattr,skew=1
#fwd=eda_pt_read_2,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=49
#fwd=eda_pt_read_4,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=getattr,skew=1
#fwd=eda_pt_read_5,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=49

fwd=eda_pt_read_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=50
fwd=eda_pt_read_2,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=50
```

```
# =====
# Primetime mix
# =====
fwd=eda_pt_mix_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=getattr,skew=1
fwd=eda_pt_mix_2,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=40
fwd=eda_pt_mix_3,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=write,skew=9
fwd=eda_pt_mix_4,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=getattr,skew=1
fwd=eda_pt_mix_5,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=40
fwd=eda_pt_mix_6,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=write,skew=9


# =====
# 100% metadata read
# =====
fwd=eda_meta_read,fsd=fs_eda_verilogsmall_2,xfersize=64k,operation=getattr




# =====
# Read throughput
# =====
fwd=eda_io_read_1,fsd=fs_eda_io_1,xfersize=1M,fileio=sequential,operation=read
fwd=eda_io_read_2,fsd=fs_eda_io_2,xfersize=1M,fileio=sequential,operation=read


# =====
# Write throughput
# =====
fwd=eda_io_write_1,fsd=fs_eda_io_1,xfersize=1M,fileio=sequential,operation=write
fwd=eda_io_write_2,fsd=fs_eda_io_2,xfersize=1M,fileio=sequential,operation=write

# ====================
```

Actual benchmark run definitions

You can run multiple benchmarks one after the other by having more than 1un definition enabled. vdbench will run the definitions in the order found in this file.

#

====================

The following run definition default includes the directio option. This will attempt to bypass the client cache which you normally will want to do in a benchmark. Otherwise, you may be testing how good your local cache is.

#

rd=default,threads=320,fwdrate=max,interval=5,pause=600,elapsed=1200,openflags=directio

The following line removes the directio flag, runs test for 20 minutes then pauses 10 minutes. The reporting interval is 5 seconds.

rd=default,threads=48,fwdrate=max,interval=2,pause=60,elapsed=600

=====

Test run file create definitions

#

Note: Uncomment both 'rd' for each profile to start fresh. The first 'rd' will delete any existing files and the second one will create/update the file system to the correct configuration. The run definitions below only delete or create files.

=====

Create Verilog file set

rd=verilog_create_clean,fwd=eda_verilog_*,format=(clean,only)
rd=verilog_create,fwd=eda_verilog_*,format=(restart,only)

Create Verilog SMALL file set

rd=verilog_s_create_clean,fwd=eda_verilogsmall_1,format=(clean,only)
rd=verilog_s_create_clean,fwd=eda_verilogsmall_6,format=(clean,only)
rd=verilog_s_create,fwd=eda_verilogsmall_1,format=(restart,only)
rd=verilog_s_create,fwd=eda_verilogsmall_6,format=(restart,only)

Create Primetime file set

rd=pt_create_clean_1,fwd=eda_pt_read_1,format=(clean,only)
rd=pt_create_clean_2,fwd=eda_pt_read_4,format=(clean,only)
rd=pt_create_1,fwd=eda_pt_read_1,format=(restart,only)
rd=pt_create_2,fwd=eda_pt_read_4,format=(restart,only)

Create throughput file set

rd=throughput_create_clean,fwd=eda_io_read_1,format=(clean,only)
rd=throughput_create_clean,fwd=eda_io_read_2,format=(clean,only)
rd=throughput_create,fwd=eda_io_read_1,format=(restart,only)
rd=throughput_create,fwd=eda_io_read_2,format=(restart,only)

Create EDA mix file set

rd=eda_create_clean,fwd=eda_mix_1,format=(clean,only)
rd=eda_create_clean,fwd=eda_mix_7,format=(clean,only)
rd=eda_create,fwd=eda_mix_1,format=(restart,only)
rd=eda_create,fwd=eda_mix_7,format=(restart,only)
=====

Test run definitions

#
Uncomment run definitions below to perform a specific benchmark run.
#
=====
Test verilog small

Definition which tries to find the maximum performance.

#rd=eda_verilogsmall_run,fwd=eda_verilogsmall_*,format=restart,fwdrate=max,elapsed=180
#rd=nitro_eda_verilogsmall_run,fwd=eda_verilogsmall_*,format=restart,fwdrate=(60000-120000,20000),threads=128,elapsed=180
#rd=nitro_eda_verilogsmall_run,fwd=eda_verilogsmall_*,format=restart,fwdrate=(l20000-200000,20000),threads=128,elapsed=180

Test verilog

Definition which tries to find the maximum performance.
#rd=eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(140000-200000,20000),elapsed=180
#rd=nitro_eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(60000-120000,20000),threads=128,elapsed=180
#rd=nitro_eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(l20000-200000,20000),threads=128,elapsed=180

Test PT large file (100GB files)

Definition which tries to find the maximum performance.

rd=nitro_eda_pt_read,fwd=eda_pt_read*,format=restart,fwdrate=600000,xfersize=1m,threads=96
#rd=nitro_eda_pt_write,fwd=eda_pt_write*,format=restart,fwdrate=60000,xfersize=1m

#rd=nitro_eda_pt_write,fwd=eda_pt_write*,format=restart,fwdrate=(60000-100000,10000),xfersize=1m
#rd=nitro_eda_pt_mix,fwd=eda_pt_mix*,format=restart,fwdrate=50000,xfersize=1m

Test read throughput
Definition which tries to find the maximum performance.
#rd=eda_io_read_run,fwd=eda_io_read*,format=restart,fwdrate=max,elapsed=180
#rd=nitro_eda_io_read_run,fwd=eda_io_read,format=restart,fwdrate=(5000-10000,1000)

Test read meta
Definition which tries to find the maximum performance.

#rd=eda_meta_read_run,fwd=eda_meta_read,format=restart,fwdrate=max,elapsed=180
#rd=nitro_eda_meta_read_run,fwd=eda_meta_read,format=restart,fwdrate=(200000-350000,10000)

Test Verilog mix

The Verilog mix overrides the normal threads count due to the way this profile is configured. Because the profile is using independent directories and hosts, the thread number is not a total threads across the entire workload, instead it is the number of threads per fwd. The standard configuration has 10 fwd defined per host. A threads setting of 1 would make each host have 10 threads running.

Definition which tries to find the maximum performance.

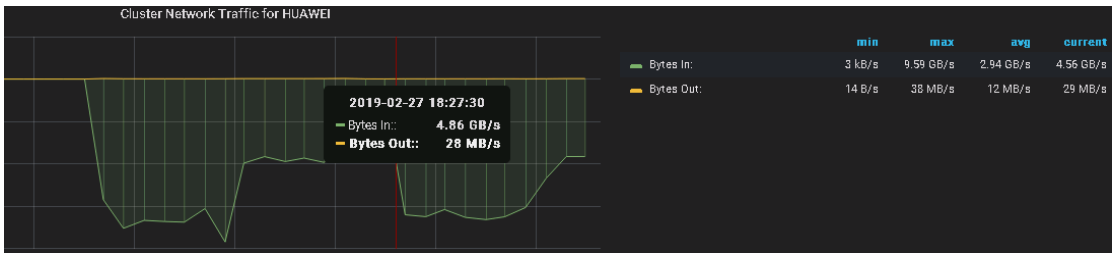
#rd=eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=max,elapsed=360,threads=1
#rd=nitro_eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(150000-500000,50000),threads=3

Test EDA mix
Definition which tries to find the maximum performance.

rd=eda_mix_run,fwd=eda_mix_*,format=restart,fwdrate=max,elapsed=360
#rd=nitro_eda_mix_run,fwd=eda_mix_*,format=restart,fwdrate=(200000-800000,50000)

Test Output and Findings

Big file generating in process. The peak writing performance is 9.5GB/s per chassis.



The highest PT write performance is at peak external network throughput rate 108Gb/s.

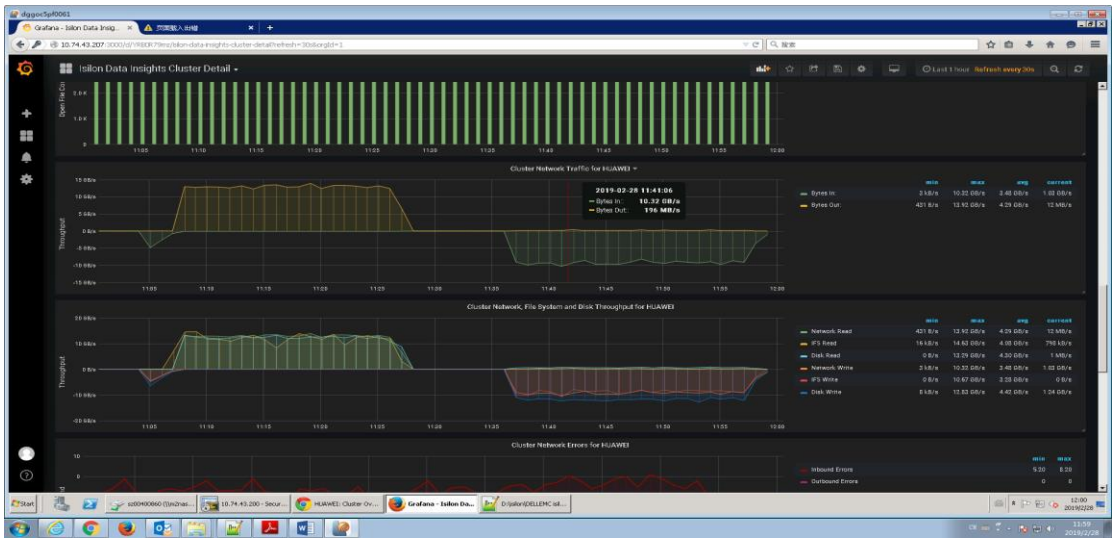
```
# Primetime like file system (100GB size large files)
# 2^4 = 16*6 files @ 95 GiB/file = 9120 GB
fsd=fs_eda_pt_large_1,anchor=/hw100/isilon/test/pt_large,shared=yes,width=2,depth=4,files=6,size=(90G,50,100G,50)
fsd=fs_eda_pt_large_2,anchor=/hw200/isilon/test/pt_large,shared=yes,width=2,depth=4,files=6,size=(90G,50,100G,50)
```

-

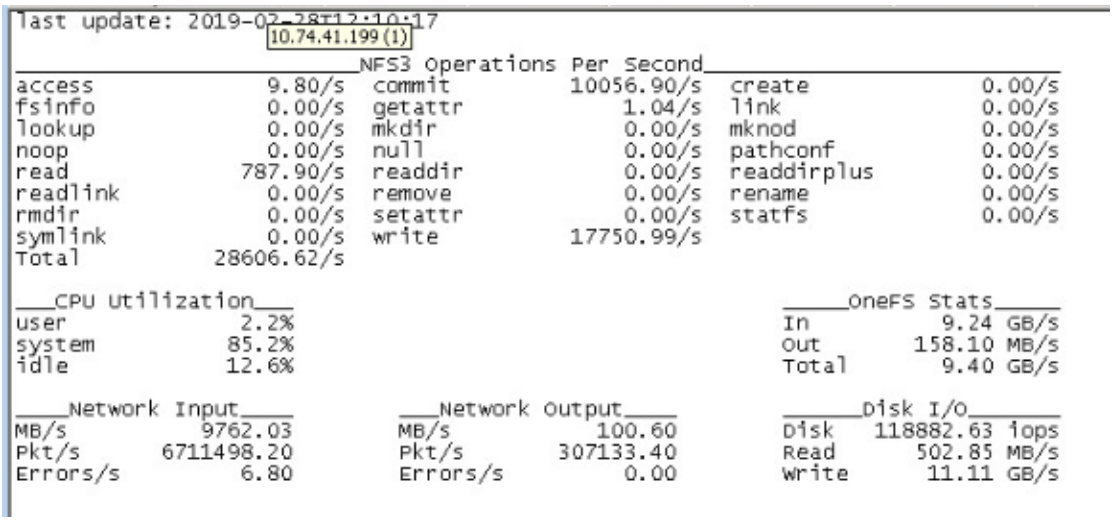
```
# Test PT large file(100GB files)
# Definition which tries to find the maximum performance
#rd=nitro_eda_pt_read,fwd=eda_pt_read*,format=restart,fwdrate=500000,xfersize=1m,threads=48
#rd=nitro_eda_pt_write,fwd=eda_pt_write*,format=restart,fwdrate=(50000-100000,10000),xfersize=1m
#rd=nitro_eda_pt_mix,fwd=eda_pt_mix*,format=restart,fwdrate=(100000-200000,10000),xfersize=1m
```

```
# =====
# Primetime 96% write
# =====
fwd=eda_pt_write_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=getattr,skew=1
fwd=eda_pt_write_2,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=1
fwd=eda_pt_write_3,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=write,skew=48
fwd=eda_pt_write_4,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=getattr,skew=1
fwd=eda_pt_write_5,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=1
fwd=eda_pt_write_6,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=write,skew=48

# =====
# Primetime 98% read
# =====
#fwd=eda_pt_read_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=getattr,skew=1
#fwd=eda_pt_read_2,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=49
#fwd=eda_pt_read_4,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=getattr,skew=1
#fwd=eda_pt_read_5,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=49

fwd=eda_pt_read_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=50
fwd=eda_pt_read_2,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=50


# =====
# Primetime mix
# =====
fwd=eda_pt_mix_1,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=getattr,skew=1
fwd=eda_pt_mix_2,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=read,skew=40
fwd=eda_pt_mix_3,fsd=fs_eda_pt_large_1,xfersize=1M,fileio=sequential,operation=write,skew=9
fwd=eda_pt_mix_4,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=getattr,skew=1
fwd=eda_pt_mix_5,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=read,skew=40
fwd=eda_pt_mix_6,fsd=fs_eda_pt_large_2,xfersize=1M,fileio=sequential,operation=write,skew=9
```

Records in Grafana

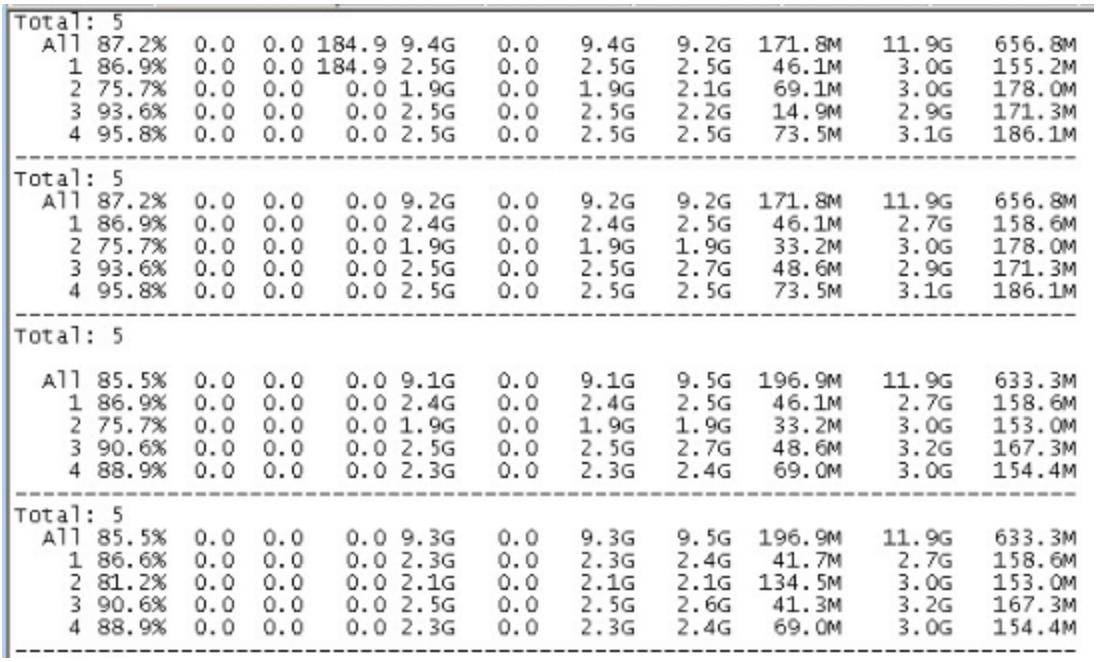


Records in OneFS CLI



```
last update: 2019-02-28T12:10:17
                                10.74.41.199 (1)
                          NFS3 Operations Per Second
access              9.80/s  commit         10056.90/s  create              0.00/s
fsinfo              0.00/s  getattr            1.04/s  link                0.00/s
lookup              0.00/s  mkdir              0.00/s  mknod               0.00/s
noop                0.00/s  null               0.00/s  pathconf            0.00/s
read              787.90/s  readdir            0.00/s  readdirplus         0.00/s
readlink            0.00/s  remove             0.00/s  rename              0.00/s
rmdir               0.00/s  setattr            0.00/s  statfs              0.00/s
symlink             0.00/s  write          17750.99/s
Total           28606.62/s

   CPU Utilization                                          OneFS Stats
user                2.2%                                In          9.24 GB/s
system             85.2%                                Out       158.10 MB/s
idle               12.6%                                Total       9.40 GB/s

    Network Input          Network Output          Disk I/O
MB/s          9762.03  MB/s          100.60  Disk   118882.63 iops
Pkt/s      6711498.20  Pkt/s      307133.40  Read      502.85 MB/s
Errors/s         6.80  Errors/s        0.00  Write      11.11 GB/s
```

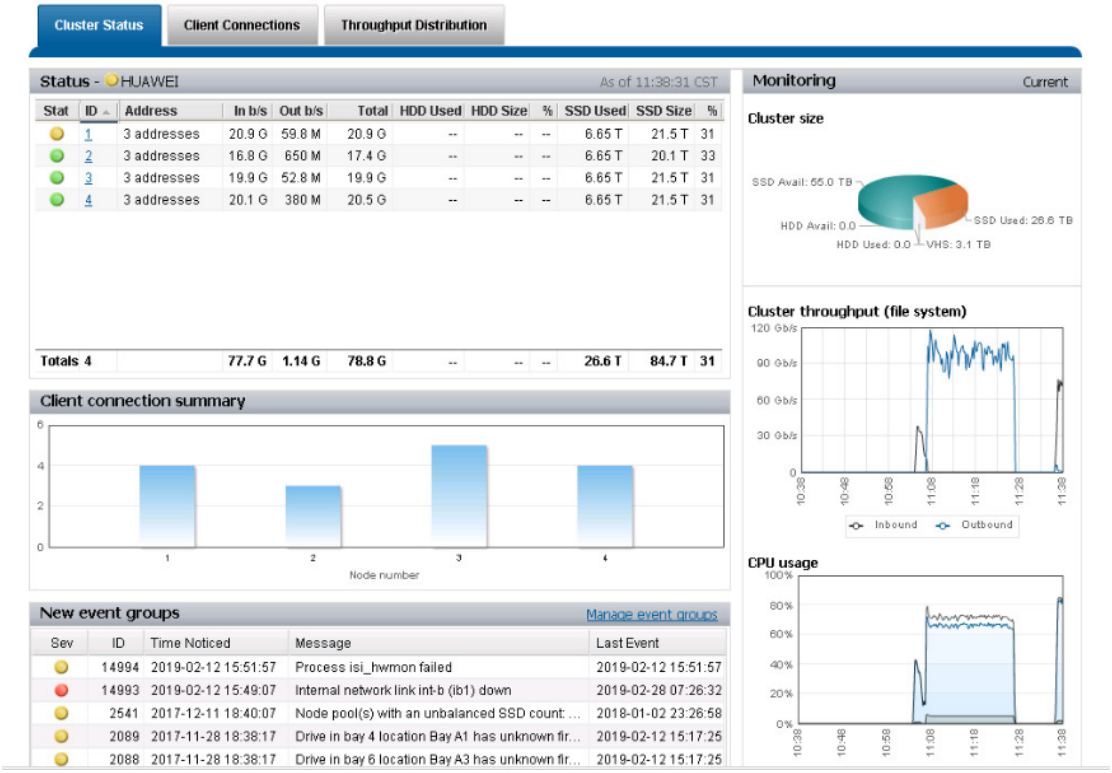
Records in OneFS CLI



```
Total: 5
  All 87.2%  0.0  0.0 184.9 9.4G  0.0  9.4G  9.2G 171.8M  11.9G  656.8M
    1 86.9%  0.0  0.0 184.9 2.5G  0.0  2.5G  2.5G  46.1M   3.0G  155.2M
    2 75.7%  0.0  0.0   0.0 1.9G  0.0  1.9G  2.1G  69.1M   3.0G  178.0M
    3 93.6%  0.0  0.0   0.0 2.5G  0.0  2.5G  2.2G  14.9M   2.9G  171.3M
    4 95.8%  0.0  0.0   0.0 2.5G  0.0  2.5G  2.5G  73.5M   3.1G  186.1M
--------------------------------------------------------------------------
Total: 5
  All 87.2%  0.0  0.0   0.0 9.2G  0.0  9.2G  9.2G 171.8M  11.9G  656.8M
    1 86.9%  0.0  0.0   0.0 2.4G  0.0  2.4G  2.5G  46.1M   2.7G  158.6M
    2 75.7%  0.0  0.0   0.0 1.9G  0.0  1.9G  1.9G  33.2M   3.0G  178.0M
    3 93.6%  0.0  0.0   0.0 2.5G  0.0  2.5G  2.7G  48.6M   2.9G  171.3M
    4 95.8%  0.0  0.0   0.0 2.5G  0.0  2.5G  2.5G  73.5M   3.1G  186.1M
--------------------------------------------------------------------------
Total: 5

  All 85.5%  0.0  0.0   0.0 9.1G  0.0  9.1G  9.5G 196.9M  11.9G  633.3M
    1 86.9%  0.0  0.0   0.0 2.4G  0.0  2.4G  2.5G  46.1M   2.7G  158.6M
    2 75.7%  0.0  0.0   0.0 1.9G  0.0  1.9G  1.9G  33.2M   3.0G  153.0M
    3 90.6%  0.0  0.0   0.0 2.5G  0.0  2.5G  2.7G  48.6M   3.2G  167.3M
    4 88.9%  0.0  0.0   0.0 2.3G  0.0  2.3G  2.4G  69.0M   3.0G  154.4M
--------------------------------------------------------------------------
Total: 5
  All 85.5%  0.0  0.0   0.0 9.3G  0.0  9.3G  9.5G 196.9M  11.9G  633.3M
    1 86.6%  0.0  0.0   0.0 2.3G  0.0  2.3G  2.4G  41.7M   2.7G  158.6M
    2 81.2%  0.0  0.0   0.0 2.1G  0.0  2.1G  2.1G 134.5M   3.0G  153.0M
    3 90.6%  0.0  0.0   0.0 2.5G  0.0  2.5G  2.6G  41.3M   3.2G  167.3M
    4 88.9%  0.0  0.0   0.0 2.3G  0.0  2.3G  2.4G  69.0M   3.0G  154.4M
--------------------------------------------------------------------------
```

CPU utilization rate

The max CPU utilization that we saw was on the full workload and registered about 80~96%.



- The Avg. protocol latency in IIQ is 1-2ms during one hour on 13rd

  The IIQ latency shows what the cluster sees, but what the client sees via vdbench is < 1 ms latency.

  Here is the explanation: the statistic method for IIQ and vdbench might vary at all, what the cluster sees is the protocol average latency, and what the vdbench software calculates is just the response time of IO from and back to the client. So it's ok, we think this is two aspect of statistic. But as a third party of benchmarking tool, the customer prefers the original output of vdbench.

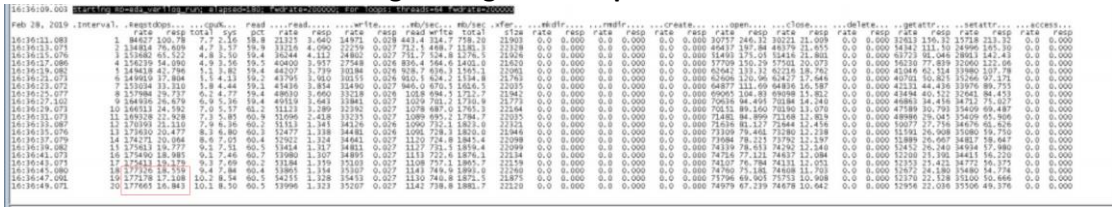
Verilog testing results

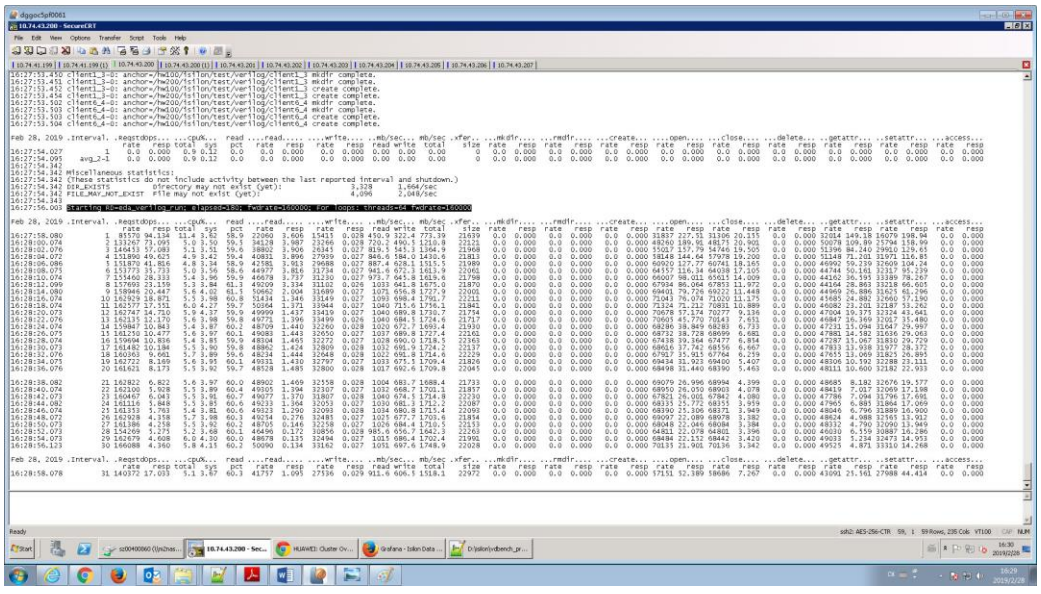
- The data set definition is below
- ```
  fsd=fs_eda_verilog_1_$host,anchor=/hw100/isilon/test/verilog/$host,width=4,depth=7,files=20,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)
  fsd=fs_eda_verilog_2_$host,anchor=/hw200/isilon/test/verilog/$host,width=4,depth=7,files=20,sizes=(1k,10,2k,10,4k,10,8k,50,16k,5,32k,5,64k,5,128k,4,1M,1)
  ```
- ```
  15:40:22.608 Anchor size: anchor=/hw200/isilon/test/verilog/client6_1: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.624 Anchor size: anchor=/hw200/isilon/test/verilog/client6_2: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.640 Anchor size: anchor=/hw200/isilon/test/verilog/client6_3: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.656 Anchor size: anchor=/hw200/isilon/test/verilog/client6_4: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.672 Anchor size: anchor=/hw200/isilon/test/verilog/client7_1: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.689 Anchor size: anchor=/hw200/isilon/test/verilog/client7_2: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.705 Anchor size: anchor=/hw200/isilon/test/verilog/client7_3: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.721 Anchor size: anchor=/hw200/isilon/test/verilog/client7_4: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.737 Anchor size: anchor=/hw200/isilon/test/verilog/client8_1: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.753 Anchor size: anchor=/hw200/isilon/test/verilog/client8_2: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.770 Anchor size: anchor=/hw200/isilon/test/verilog/client8_3: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:22.786 Anchor size: anchor=/hw200/isilon/test/verilog/client8_4: dirs:       21,844; files:      327,680; bytes:     8.009g (8,599,528,448)
  15:40:30.392 Estimated totals for all 72 anchors: dirs: 1,398,016; files: 20,971,520; bytes: 512.572g
  ```

```
# =====
# Verilog
# =====
# This workload does not used a shared file system. Each process will create its own directory
# structure based on the host. This is done by adding the host= parameter to the definition.
# This will restrict this fwd to a specific host. However becuase of this change, we cannot
# easily use the skew parameter to create work load percentages. To emulate the same behavior

# This will restrict this fwd to a specific host. However becuase of this change, we cannot
# easily use the skew parameter to create work load percentages. To emulate the same behavior
# we create multipled fwd with the same operation thus manually creating a correct workload
# mix. This method is more coarse and you can only easily do 10% or 5% skews of the workload.
# To do 10% skews you need 10 fwd lines and to do 5% skews you need 20 fwd lines
# This will also require a decrease in the number of threads to a small number like 1 or 2 as
# each host will have #fwd*#hosts*threads running. This needs to be set at the run definition
fwd=eda_verilog_1_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=getattr
fwd=eda_verilog_2_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=getattr
fwd=eda_verilog_3_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=getattr
fwd=eda_verilog_4_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=setattr
fwd=eda_verilog_5_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=setattr
fwd=eda_verilog_6_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=read
fwd=eda_verilog_7_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=read
fwd=eda_verilog_8_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=read
fwd=eda_verilog_9_$host,host=$host,fsd=fs_eda_verilog_1_$host,xfersize=64k,operation=write
fwd=eda_verilog_10_$host,host=$host,fsd=fs_eda_verilog_2_$host,xfersize=64k,operation=write
# Below is a shared workload version, similar to the verilog SMALL
#fwd=eda_verilog_1,,fsd=fs_eda_verilog,xfersize=64k,operation=getattr,skew=30
#fwd=eda_verilog_2,fsd=fs_eda_verilog,xfersize=64k,operation=access,skew=2
#fwd=eda_verilog_3,fsd=fs_eda_verilog,xfersize=64k,operation=setattr,skew=18
#fwd=eda_verilog_4,fsd=fs_eda_verilog,xfersize=64k,operation=read,skew=30
#fwd=eda_verilog_5,fsd=fs_eda_verilog,xfersize=64k,operation=write,skew=20
```

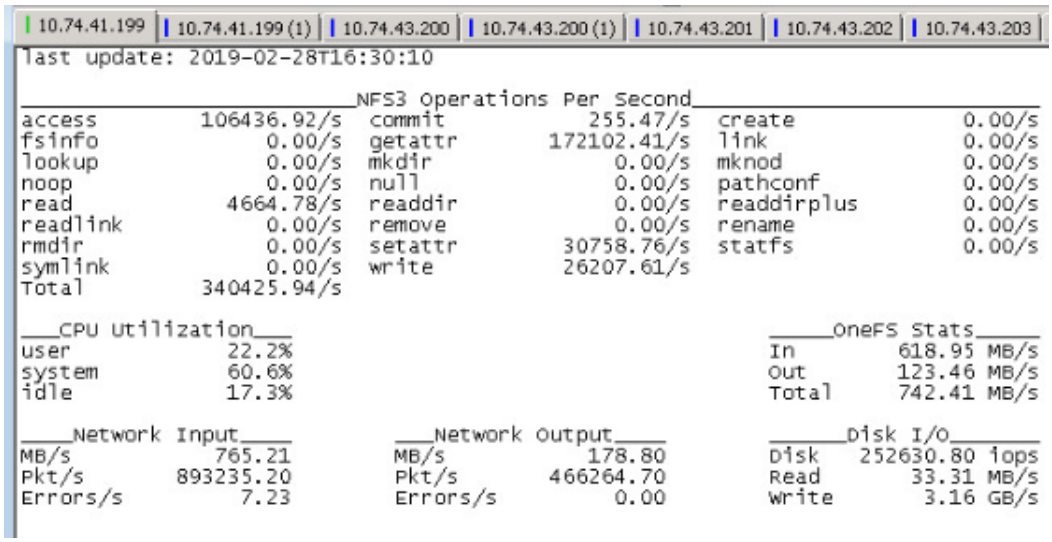
```
# =====
# Test verilog
# Definition which tries to find the maximum performance
rd=eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(140000-200000,20000),elapsed=180
#rd=nitro_eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(60000-120000,20000),threads=128,elapsed=180
#rd=nitro_eda_verilog_run,fwd=eda_verilog_*,format=restart,fwdrate=(120000-200000,20000),threads=128,elapsed=180
```

- **Vdbench's screenshot of running Verilog - 177k ops**



- 

Meanwhile, it shows 340K ops in the OneFS CLI through isi statistics.

- 

```
10.74.41.199 | 10.74.41.199 (1) | 10.74.43.200 | 10.74.43.200 (1) | 10.74.43.201 | 10.74.43.202 | 10.74.43.203
Last update: 2019-02-28T16:30:10

                              NFS3 Operations Per Second
access        106436.92/s  commit          255.47/s  create          0.00/s
fsinfo             0.00/s  getattr      172102.41/s  link            0.00/s
lookup             0.00/s  mkdir             0.00/s  mknod           0.00/s
noop               0.00/s  null              0.00/s  pathconf        0.00/s
read            4664.78/s  readdir           0.00/s  readdirplus     0.00/s
readlink           0.00/s  remove            0.00/s  rename          0.00/s
rmdir              0.00/s  setattr       30758.76/s  statfs          0.00/s
symlink            0.00/s  write         26207.61/s
Total         340425.94/s

   CPU Utilization                                        OneFS Stats
user              22.2%                                  In        618.95 MB/s
system            60.6%                                  Out       123.46 MB/s
idle              17.3%                                  Total     742.41 MB/s

    Network Input          Network Output          Disk I/O
MB/s          765.21   MB/s          178.80   Disk   252630.80 iops
Pkt/s      893235.20   Pkt/s      466264.70   Read       33.31 MB/s
Errors/s        7.23   Errors/s        0.00   Write       3.16 GB/s
```

F800's testing performance results summary

	Infinity F800				
	Cluster ops	**ops per node**	**Cluster throughput(MB/s)**	**(MB/s) per node**	**Latency (ms)**
Verilog	177665	44416	N/A	N/A	10
Primetime	N/A	N/A	9500	2300	1.308
BigFileGenerating			10670		
Sequential Read			14360		

Post-Evaluation Conclusion

- The 1-chasis F800 system was tested and Isilon performed very well in PrimeTime parallel write scenario.
- It is referenceable to simulate the customer's actual EDA environment by using vdbench software, especially in Verilog and PrimeTime.
- The F800's big block parallel write performance 2.3GB/s@2ms per node for PT is ok for the acceptance of the customer.
- The OPS numbers for Verilog and the EDA mix look a little on the low side. Especially for Verilog. we could achieve at least twice that number, but we need more clients/node. It may be that vdbench has some limitations or some peculiarities that we still have to work out to improve the performance. The second item is that we have stock Linux clients. No performance tuning of the TCP stack has been done. One point to note during the testing is that neither the clients nor Nitro nodes had very high CPU utilization, suggesting there is more performance available.