Improved I/O Using Native Spectrum Scale (GPFS) Clients on a Cray XC System

Jesse Hanley
Oak Ridge National Laboratory
Oak Ridge, TN
hanleyja@ornl.gov

Chris Muzyn
Oak Ridge National Laboratory
Oak Ridge, TN
muzyncj@ornl.gov

Matt Ezell
Oak Ridge National Laboratory
Oak Ridge, TN
ezellma@ornl.gov

I. Abstract

The National Center for Computational Sciences (NCCS) created a method for natively routing communication between Cumulus, a Cray Rhine/Redwood XC40 supercomputer deployed for DOE's Atmospheric Radiation Measurement (ARM) facility, and Wolf, a Spectrum Scale file system, using IP over InfiniBand (IPoIB) and native Linux kernel tools. Spectrum Scale lacks a routing facility like Lustre's LNET capability. To facilitate communication between storage and compute, Cumulus originally projected Wolf using Cray's Data Virtualization Service (DVS). The lack of native file system support impacted users as they ported workflows to Cumulus. To support these and future use cases, the DVS projection method has been replaced by a native Spectrum Scale cluster on Cumulus that routes traffic to and from Wolf at comparable performance. This paper presents an introduction to the systems involved, a summary of motivations for the work, challenges faced, and details about the native routing configuration.

II. Introduction

Cumulus, a 112 node Cray Rhine/Redwood XC40 system, is a system deployed for the Atmospheric Radiation Measurement (ARM) project, sponsored by the US Department of Energy. Cumulus shares access to Wolf, a center-wide Spectrum Scale (GPFS) file system composed of Dell R630 servers, with dual-socket Broadwell processors and backed by DDN Storage 14KX block storage. Wolf provides ~7.7 Petabytes of storage for the low-security zone within the National Center for Computational Sciences (NCCS). Six Cumulus nodes, designated for external I/O, connect to a central EDR Mellanox InfiniBand switch. These nodes only have FDR adapters.

Historically, NCCS has routed file system traffic through a number of designated nodes within a given compute cluster. This allows for scalable access to shared parallel file system resources. Clustered file systems like Lustre and Spectrum Scale (GPFS) provide benefits to HPC workloads that traditional file systems lack. The Lustre Networking (LNet) protocol can be configured to forward Lustre traffic through one or more nodes, allowing for communication between distinct network fabrics. This enables an internal network fabric, like Cray's Aries interconnect, to communicate with storage servers on a separate fabric. Unfortunately, GPFS lacks such a facility. To enable access to the Wolf file system from Cumulus' compute nodes, NCCS originally deployed Cumulus with six I/O nodes. These nodes were designated as clients to the file system and used Cray's Data Virtualization Service (DVS) to project Wolf to the compute nodes.

After deployment, there was some disappointment in the initial performance between Cumulus and Wolf. This challenged the center to redefine how the XC platform communicates with a GPFS file system to overcome lackluster interaction and POSIX issues.

By utilizing routing and network configurations available in modern kernels, staff can better support the mix of streaming I/O, interactive use, and small file I/O present on Cumulus. Additionally, the compute cluster can now leverage the benefits of the GPFS client code. This transition allows for fewer user code changes, supporting advantageous features of GPFS, like byte-range level locking. In this paper, we show the design, implementation details, and trade-offs to deploying this method of file system access. Site-specific information, such as internal IP addresses, has been removed.

A. *The Problem*

Cray XC40 machines rely on the external InfiniBand ports of the service nodes to perform any kind of external I/O to the machine. As the compute nodes only have access to the internal Aries network, the issue being addressed is how can the gap between the InfiniBand network and the internal Aries network be bridged.

A secondary problem that arises is where to position the remote GPFS cluster. A GPFS cluster such as Wolf is composed of nodes referred to as Network Shared Disk (NSD) Servers. Mounting the file system created from these NSD servers requires nodes to be GPFS clients. This is accomplished by granting access to remote GPFS clusters. A remote GPFS cluster is used to isolate management domains, as well as, to protect the file system cluster from problematic clients. The position of the GPFS cluster affects the users interface to the the file system.

B. *Challenges of Existing Solution*

The existing solution bridges the gap between the networks by utilizing Cray's DVS in order to forward a mount of Wolf from the service nodes to the internal compute nodes. In this solution, the I/O nodes form the remote GPFS cluster and mount the GPFS filesystem from the NSD servers over the InfiniBand network. Several issues were found with the DVS projection solution. First, cache coherency becomes an issue, as DVS relies on different modes that are logic based in order to offer atomicity rather than providing cache coherence. Second, while DVS may be a convenient way to leverage the LNET framework to route a file system that does not have native routing, it lacks features that many users at NCCS expect. For instance, the 'flock' system call is unavailable on a file system projected by DVS, which caused issues with user software libraries. Furthermore, when a GPFS file system is projected by DVS, users lose the ability to perform byte-level locking on files. In regards to performance, while the DVS solution excels in large I/O situations because of caching mechanisms, any use non-large I/O patterns will return sub optimal performance.

C. *Benchmarking*

1) mdtest: The mdtest benchmark, now part of the IOR benchmark repository, provides an extensive tool to simulate and measure file system performance. It excels at I/O that results in metadata-intensive workloads. For the benchmarking referenced within this paper, staff used mdtest version 1.9.3 and varied the number of processes, number of hosts, and the amount of data written and read to see how these combinations would scale.

2) IOR: The IOR benchmark is a frequently used measure of file system performance. It is heavily tunable and can be used to demonstrate a variety of workloads. To simulate traditional HPC file-per-process workloads that run on Cumulus and other machines within NCCS, staff measured how IOR performed at various node counts, as well as the number of tasks per node. Staff performed all IOR benchmarking using IOR version 2.10.3 against the POSIX interface, along with the following parameters:

- `-F` to ensure that each process accessed a separate file
- `-i 3` to perform three iterations of the test
- `-b 10240g` to ensure sufficient data size
- `-t 4m` to perform operations that could be aligned with Wolf's GPFS blocksize
- `-C` to reorder tasks during readback and minimize client caching
- `-g` to use barriers between the phases of the IOR run
- `-e` to perform an fsync after POSIX write close
- `-D` to stonewall I/O

III. Linux kernel routing

As part of the Linux kernel, network routing is core to how machines communicate. These routes contain information about how to reach network addresses, and are stored in the routing policy database, also know as the RPDB. During the benchmarking presented in this paper, Cumulus and Wolf ran 3.12 and 3.10 versions of the kernel, respectively. Linux kernel version 2.6.39 introduced a new, more efficient way to store routes. In this data structure, routes are split across three tables: `local`, `main`, and `default`. The kernel manages entries with the `local` table; these are for local addresses, such as the route to the loopback device. The `main` table generally contains the routes created by users. If a default route is configured, it will be listed within the `main` table. The `default` table is usually empty, but can be configured to route packets not routed by other rules. As of Linux kernel version 4.1, these three tables are collapsed into one.

When configured, the kernel can also support user-defined routing tables. Then, using tools like `ip`, a policy can be defined. The kernel will then check that policy against packets to determine how it should be routed. The Linux kernel also has a feature to enable multiple paths to a destination, allowing administrators to permit any one of several nodes to satisfy a routing requirement.

IV. Implementation

A shared EDR InfiniBand switch facilitates communication between Cumulus and Wolf, represented by a simplified version in *Figure 1.* In addition to the Wolf NSD servers, and the six Cumulus I/O servers, there are other miscellaneous machines on this fabric, including high-speed data transfer nodes.

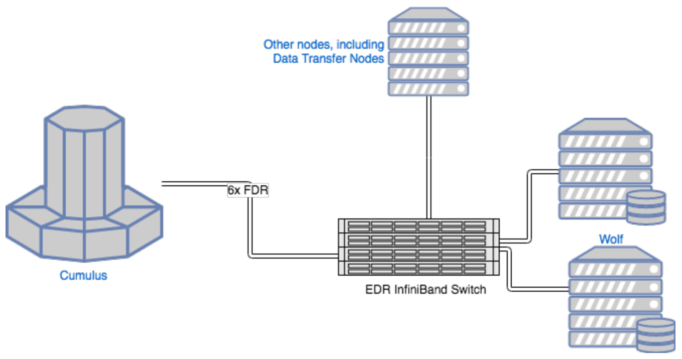

Figure 1. NCCS Open Diagram

A. *Kernel Features*

To use the advanced routing features described above, the kernel will need to be complied with the following configuration flags:

- CONFIG_IP_ROUTE_MULTIPATH=y
- CONFIG_IP_MULTIPLE_TABLES=y

The kernel Wolf ran during this configuration, 3.10 based from Red Hat Enterprise Linux 7.4, had these options enabled. The kernel used for login and service nodes within Cumulus also had these features enabled by default.

B. Networking setup

The Aries network of Cumulus resides on a private /16 network, referenced as 10.100.0.0/16 for the purposes of this paper. The IPoIB network for the InfiniBand fabric is 10.10.0.0/24. The six I/O service nodes (sn) are dual-homed on these networks with network addresses documented in *Table 1*.

Table I

Host	Aries Address	IPoIB Address
sn1	10.100.0.91	10.10.0.11
sn2	10.100.0.92	10.10.0.12
sn3	10.100.0.93	10.10.0.13
sn4	10.100.0.94	10.10.0.14
sn5	10.100.0.95	10.10.0.15
sn6	10.100.0.96	10.10.0.16

1) Storage servers: For the Wolf servers, the advanced routing configuration was accomplished with the `NetworkManager-dispatcher-routing-rules`, available in the `optional` RedHat Enterprise Linux repository. We created configuration files to configure these networks on boot. These rules define which table to use based on the source address. For the six I/O nodes in Cumulus, the configuration is like so:

```
# /etc/sysconfig/network-scripts/rule-interface
from 10.10.0.11 table 1
from 10.10.0.12 table 2
from 10.10.0.13 table 3
from 10.10.0.14 table 4
from 10.10.0.15 table 5
from 10.10.0.16 table 6
```

For each of these lookup tables, we define a default route that ensures response traffic on the IPoIB network uses the same I/O node. Finally, each of the I/O nodes are treated as a potential network for multipath routing to the compute nodes. This is achieved by the `nexthop` directive in the configuration:

```
# /etc/sysconfig/network-scripts/route-interface

10.100.0.0/16 dev ib1 scope link table 1
default via 10.10.1.91 table 1
10.100.0.0/16 dev ib1 scope link table 2
default via 10.10.1.92 table 2
10.100.0.0/16 dev ib1 scope link table 3
default via 10.10.1.93 table 3
10.100.0.0/16 dev ib1 scope link table 4
default via 10.10.1.94 table 4
10.100.0.0/16 dev ib1 scope link table 5
default via 10.10.1.95 table 5
10.100.0.0/16 dev ib1 scope link table 6
default via 10.10.1.96 table 6
10.100.0.0/16 scope global \
  nexthop dev ib1 via 10.10.0.11 \
  nexthop dev ib1 via 10.10.0.12 \
  nexthop dev ib1 via 10.10.0.13 \
  nexthop dev ib1 via 10.10.0.14 \
  nexthop dev ib1 via 10.10.0.15 \
  nexthop dev ib1 via 10.10.0.16
```

Though routes can be weighted differently, the default equal-priority works well for the deployment at NCCS. Information about the route selection can be tracked by the kernel. If configured with the `CONFIG_IP_FIB_TRIE_STATS` kernel config option, information about the internal trie structures is available through procfs by viewing the `/proc/net/fib_triestat` file. This kernel config option is enabled by default on stock RHEL 7 installations.

2) Compute: Cumulus has six service nodes that are connected to the storage network via IPoIB. IP forwarding has been enabled on the service nodes in order for traffic to pass from the compute nodes to the storage network. Since the service nodes no longer mount the GPFS file system, they no longer form a GPFS cluster. Instead, every compute and login node in the system will now form a single encompassing GPFS cluster. In order for the compute nodes to join the cluster, changes in the options of the Cray compute kernel were made. The options pertain to Advanced Routing features that are enabled on the service node kernels by default. Cray has been asked to include the following options as a default in their compute kernels going forward.

```
CONFIG_IP_MULTICAST=y
CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_IP_MULTIPLE_TABLES=y
CONFIG_IP_ROUTE_MULTIPATH=y
CONFIG_IP_ROUTE_VERBOSE=y
```

Once these kernel options have been compiled into the running kernel, routing tables similar to those on Wolf can be used to route traffic through the service nodes to the compute network. There exists one table per service node in the system. These tables are created upon each boot via an Ansible play. Note that the “iproute2” package must be installed from the Cray provided SLES repositories in order to set and view tables.

```
# ip route show table all
default via 10.100.0.91 dev ipogif0  table 1
10.10.1.0/24 dev ipogif0  table 1  scope link
default via 10.100.0.92 dev ipogif0  table 2
10.10.1.0/24 dev ipogif0  table 2  scope link
```

```
default via 10.100.0.93 dev ipogif0  table 3
10.10.1.0/24 dev ipogif0  table 3  scope link
default via 10.100.0.94 dev ipogif0  table 4
10.10.1.0/24 dev ipogif0  table 4  scope link
default via 10.100.0.95 dev ipogif0  table 5
10.10.1.0/24 dev ipogif0  table 5  scope link
default via 10.100.0.96 dev ipogif0  table 6
10.10.1.0/24 dev ipogif0  table 6  scope link
10.10.1.0/24
  nexthop via 10.100.0.91  dev ipogif0 weight 1
  nexthop via 10.100.0.92  dev ipogif0 weight 1
  nexthop via 10.100.0.93  dev ipogif0 weight 1
  nexthop via 10.100.0.94  dev ipogif0 weight 1
  nexthop via 10.100.0.95  dev ipogif0 weight 1
  nexthop via 10.100.0.96  dev ipogif0 weight 1
10.100.0.0/16 dev ipogif0  proto kernel
  scope link  src 10.100.0.1
broadcast 10.100.0.0 dev ipogif0  table
  local  proto kernel  scope link  src 10.100.0.1
local 10.100.0.1 dev ipogif0  table local
  proto kernel  scope host  src 10.100.0.1
broadcast 10.100.255.255 dev ipogif0  table
  local  proto kernel  scope link  src 10.100.0.1
broadcast 127.0.0.0 dev lo  table local
  proto kernel  scope link  src 127.0.0.1
local 127.0.0.0/8 dev lo  table local
  proto kernel  scope host  src 127.0.0.1
local 127.0.0.1 dev lo  table local
  proto kernel  scope host  src 127.0.0.1
broadcast 127.255.255.255 dev lo  table local
  proto kernel  scope link  src 127.0.0.1
```

The six service nodes required the use of a persistent /var/mmfs/ directory, which is the same for the compute and login nodes. Once the directory was made persistent via the Cray Configurator, the GPFS GPL layer was able to be compiled.

C. Tuning

1) Storage servers: Due to the reliance on InfiniBand over IP (IPoIB), the networking layer has to be tuned for optimal performance. This is typically achieved by increasing more memory to the corresponding network buffers. The majority of tunings applied come from both site experience tuning networks for data transfer, as well as tunings from ESnet and Mellanox.

Most of the tcp tuning deployed is heavily documented in guides for high-bandwidth Ethernet networks. Since IPoIB relies on some of the same principles, the tuning carries over well. For the benchmarks discussed in this paper, the Wolf NSD servers were configured with the following `sysctl` tunings:

```
  net.ipv4.tcp_timestamps=0
  net.ipv4.tcp_sack=1
  net.ipv4.tcp_low_latency=1
  net.ipv4.tcp_adv_win_scale=1
  net.core.netdev_max_backlog=250000
  net.core.rmem_max=536870912
  net.core.wmem_max=536870912
  net.core.optmem_max=536870912
  net.ipv4.tcp_mem=4096 87380 268435456
  net.ipv4.tcp_rmem=4096 87380 268435456
  net.ipv4.tcp_wmem=4096 87380 268435456
  net.ipv4.tcp_no_metrics_save=1
  net.core.default_qdisc=fq
  net.ipv4.tcp_congestion_control=htcp
  net.ipv4.tcp_mtu_probing=1
  net.ipv4.tcp_fin_timeout=30
  net.ipv4.tcp_tw_recycle=1
  net.ipv4.tcp_tw_reuse=1
```

One of the most influential tunings was the change from datagram mode to connected mode on the InfiniBand interfaces. By switching to connected mode, the IPoIB interface can operate at a larger MTU, increasing the amount of traffic transferred per packet. Additionally, we increased the `txqueuelen` on the InfiniBand interfaces to 10000.

2) Compute Cluster GPFS: The Cumulus compute nodes received similiar tuning. They were less aggressively tuned in order to reduce impact on users.

D. Performance Changes

1) Single client performance: One of the largest changes observed from this deployment is to single client performance. With a native Spectrum Scale (GPFS) client, compute nodes saw an increase in streaming read speeds. *Table II* shows the benchmarking results from a DVS projected file system, while *Table III* shows results after moving to a Native Spectrum Scale (GPFS) client. There is also a slight improvement to streaming write speeds with I/O threads. These changes are visualized in *Figure 2*.

Table II

Task per node	Write (MB/s)	Read (MB/s)
1	3053.26	2008.49
2	4531.97	2660.09
4	5617.79	2759.58
8	5504.65	3654.27

Table III

Task per node	Write (MB/s)	Read (MB/s)
1	2912.81	4029.48
2	5229.82	7593.69
4	6213.73	9358.46
8	6303.60	8867.00

In addition to some streaming I/O benefit, there are large improvements to metadata operations. File creation, stat, and removal rates saw signficant improvements when running with a native Spectrum Scale (GPFS) client. *Table IV* and *Table V* show metadata benchmarks performed with *mdtest*, including writing and reading various amounts of data to a set of ten-thousand files. Only the file creation, deletion and stat rates are recorded in these tables.

Table IV

Tasks	Write Data	Read Data	Creates	Stat	Removals
1	0	0	321	13472	225
1	1024	1024	239	13250	242
1	4096	4096	157	13066	227
1	32768	32768	155	12918	232
2	0	0	906	23808	425
2	1024	1024	618	22988	530
2	4096	4096	350	22168	533
2	32768	32768	350	22316	535
4	0	0	2039	45016	925
4	1024	1024	1363	38862	1160
4	4096	4096	771	35991	923
4	32768	32768	755	35215	817
6	0	0	3336	67625	1028
6	1024	1024	2095	45785	1040
6	4096	4096	1160	39224	1298
6	32768	32768	1149	39052	1000
8	0	0	4202	80737	1336
8	1024	1024	2894	72797	1261
8	4096	4096	1512	46889	1238
8	32768	32768	1513	49482	1153

Table V

Tasks	Write Data	Read Data	Creates	Stat	Removals
1	0	0	13241	279038	20418
1	1024	1024	1179	286249	20827
1	4096	4096	715	284019	20728
1	32768	32768	709	287963	20585
2	0	0	26915	427140	39281
2	1024	1024	2497	434452	37055
2	4096	4096	1522	432041	37154
2	32768	32768	1496	435386	36666
4	0	0	30320	666998	68725
4	1024	1024	4997	730824	68547
4	4096	4096	2934	724235	60700
4	32768	32768	2906	719860	56447
6	0	0	16330	1014392	65745
6	1024	1024	7520	1009505	70702
6	4096	4096	4036	1012704	66587
6	32768	32768	3995	1004775	66093
8	0	0	12493	1269373	73599
8	1024	1024	9664	1268438	76963
8	4096	4096	5179	1271087	77932
8	32768	32768	5238	1270283	77263

Figure 3, *Figure 4*, *Figure 5*, and *Figure 6* show single client performance comparisons between a DVS projected version of Wolf and a native client mount for file sizes of 0 bytes, 1024 bytes, 4096 byte, and 32768 bytes. The number of file stats performed per second is omitted from the figures. The native client performs the action so quickly for the number of test files that the rate skews the other results when visualized.

2) Multi-client performance: Scaled I/O across multiple clients shows a potential bottleneck in the native cluster

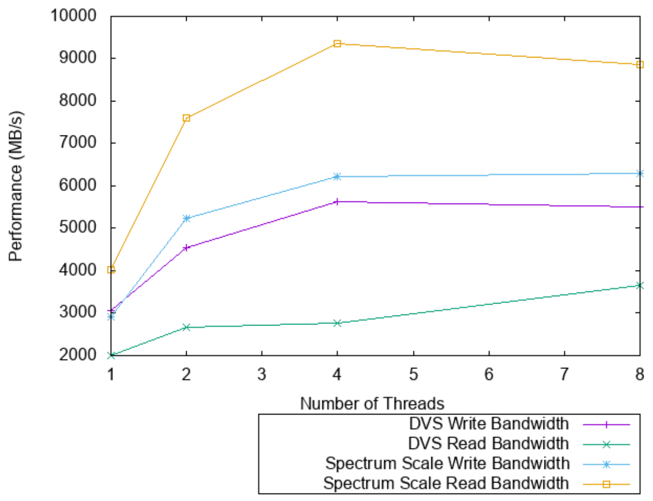

Figure 2. Single Client Streaming Performance

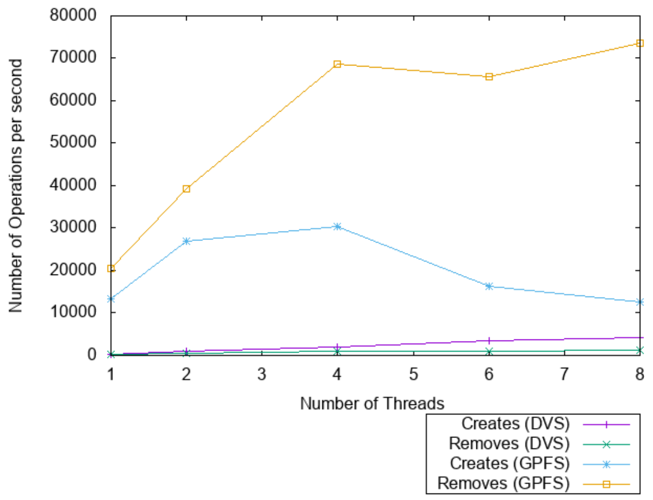

Figure 3. Metadata Performance (0 length files)

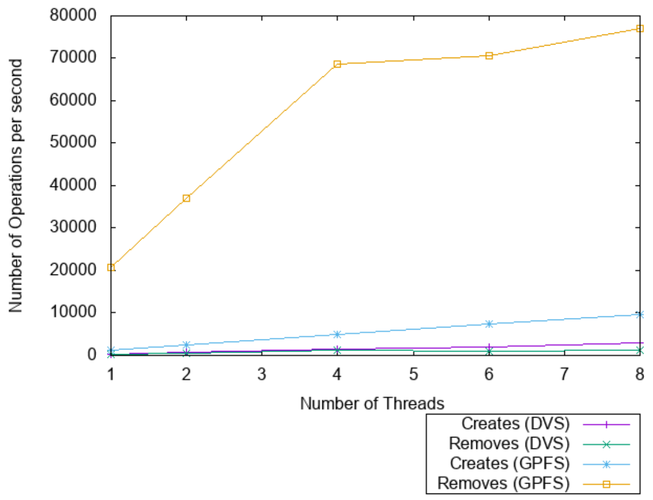

Figure 4. Metadata Performance (1024 byte length files)

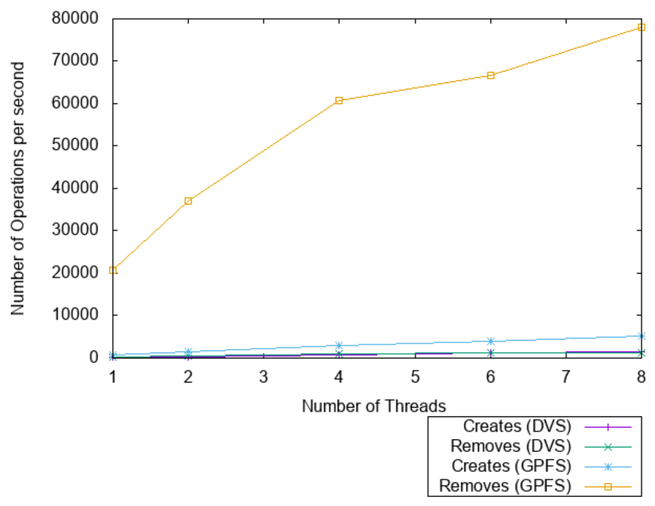

Figure 5. Metadata Performance (4096 byte length files)

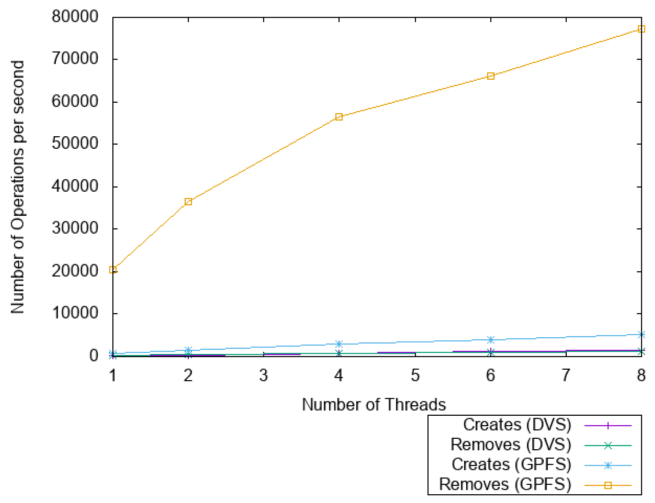

Figure 6. Metadata Performance (32768 byte length files)

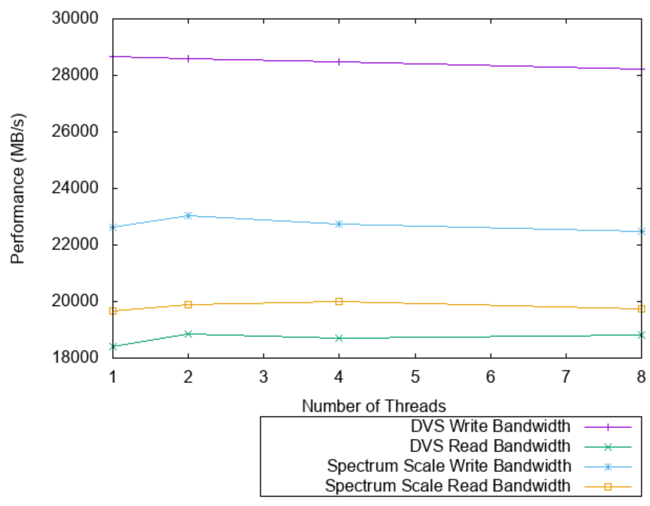

Figure 7. Streaming Performance - 16 clients

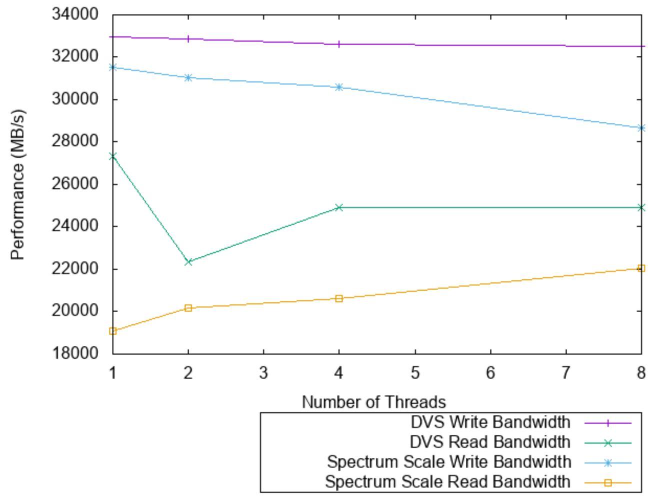

Figure 8. Streaming Performance- 64 clients

method. Streaming I/O does not scale as well when using a native client. In our testing, the DVS projected file system reached a higher peak performance, and was able to do so with a small portion of the total number of compute nodes. *Figure 7* and *Figure 8* show the results of streaming I/O benchmarks across 16 and 64 nodes, respectively.

Though single client performance is improved with a native mount, the total bandwidth to and from the file system is reduced.

However, metadata operations do show much better scaling with the native client than with the DVS projected file system. Results from these benchmarking jobs are summarized in *Table VI* and *Table VII* for the DVS projected file system and the native mount, respectively. The number of tasks listed refers to the number of processes on each of the 64 compute nodes involved in the profiling. Though the *mdtest* benchmark provides benchmarks on a number of metadata actions, only statistics for file creates, stats, and removals are recorded in these tables.

Table VI

Tasks	Write Data	Read Data	Creates	Stat	Removals
1	0	0	21802	379091	9567
1	1024	1024	13042	71530	9354
1	4096	4096	6364	340077	9527
1	32768	32768	5160	239251	8881
2	0	0	10806	319709	11596
2	1024	1024	11724	64701	11649
2	4096	4096	6812	318545	11593
2	32768	32768	7130	345619	11165
4	0	0	11699	301274	13761
4	1024	1024	10732	58568	13811
4	4096	4096	7102	290813	13695
4	32768	32768	6552	285046	13327
6	0	0	11617	274456	14572
6	1024	1024	10597	61514	14180
6	4096	4096	6531	267940	14421
6	32768	32768	6804	265482	14398
8	0	0	10749	237668	16069
8	1024	1024	10338	55703	15703
8	4096	4096	6015	144336	15633
8	32768	32768	6198	225810	16044

Table VII

Tasks	Write Data	Read Data	Creates	Stat	Removals
1	0	0	228868	52781	45725
1	1024	1024	58098	84120	43698
1	4096	4096	32064	53672	43178
1	32768	32768	32379	56032	46770
2	0	0	417835	90642	85032
2	1024	1024	74883	113421	86150
2	4096	4096	42558	100563	89861
2	32768	32768	42682	105663	90761
4	0	0	433489	100419	83624
4	1024	1024	117758	110989	84836
4	4096	4096	47252	103272	88121
4	32768	32768	50155	107880	89048
6	0	0	327876	102583	89003
6	1024	1024	146164	111844	90773
6	4096	4096	53678	104975	90644
6	32768	32768	53676	107105	91578
8	0	0	357877	105872	94807
8	1024	1024	162445	110641	92390
8	4096	4096	51000	111943	95742
8	32768	32768	50279	106426	90773

V. Tradeoffs

Due to the increase in GPFS cluster size when using the Native GPFS method, more settings can affect performance. Now that the remote GPFS cluster is running on the compute and login nodes, the GPFS client cache consumes memory on all of these nodes. Because the login nodes have less RAM than the compute nodes, this cache must remain small across all nodes. Cumulus does not typically see large amounts of concurrent users on its 32 GB login nodes, therefore their cache set to 8 GB.

Using the native GPFS method introduces the potential for jitter on file system access. This jitter can have multiple causes such as coalesced I/O, as the cluster tries to send larger messages to the file system, and communication among other nodes for cache coherency.

VI. Future work

The work presented here will continue to evolve. Over the next year, we plan to upgrade to Spectrum Scale version 5 on both the Cumulus cluster and the Wolf cluster. We also plan to devote more resources into pinpointing and, if possible, eliminating the bottleneck or overhead that is reducing the maximum throughput of streaming I/O while using the native client.

Resources

[1] R. Rosen, *Linux kernel networking: Implementation and theory.* Apress, 2014.

[2] B. Hubert, *Linux advanced routing & traffic control howto.* s.n., 2010.

[3] S. Lowe, "A quick introduction to linux policy routing - scott's weblog - the weblog of an it pro specializing in cloud computing, open source, networking, and virtualization," *Scott's Weblog.* Scott Lowe, May-2013 [Online]. Available: https://blog.scottlowe.org/2013/05/29/a-quick-introduction-to-linux-policy-routing/

[4] S. Sugiyama and D. Wallace, "Cray dvs: Data virtualization service," *Cray DVS: Data Virtualization Service*, 2008 [Online]. Available: https://cug.org/5-publications/proceedings_attendee_lists/2008CD/S08_Proceedings/pages/Authors/16-19Thursday/Wallace-Thursday16B/Sugiyama-Wallace-Thursday16B-paper.pdf

[5] IBM Systems; Technology Group, Aug-2012.

[6] "4.2. Routing for multiple uplinks/providers," *Routing for multiple uplinks/providers.* [Online]. Available: http://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.rpdb.multiple-links.html

[7] S. Oral *et al.*, "OLCF's 1 tb/s, next-generation lustre file system.".

[8] "Why use lustre - hpdd community space," *HPDD Community Wiki.* Intel [Online]. Available: https://wiki.hpdd.intel.com/display/PUB/Why Use Lustre

[9] "Product overview," *IBM Knowledge Center.* [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/STXKQY_5.0.0/com.ibm.spectrum.scale.v5r00.doc/bl1in_IntroducingIBMSpectrumScale.htm

[10] Y. Cotronis and J. J. Dongarra, *Recent advances in parallel virtual machine and message passing interface: 8th european pvm/mpi users group meeting, santorini/thera, greece, september 23-26, 2001: Proceedings.* Springer, 2001.

[11] *4.8. Routing Tables.* [Online]. Available: http://linux-ip.net/html/routing-tables.html

[12] V. Bernat, "IPv4 route lookup on linux," *IPv4 route lookup on Linux / Vincent Bernat.* Jun-2017 [Online]. Available: https://vincent.bernat.im/en/blog/2017-ipv4-route-lookup-linux#lookup-with-a-level-compressed-trie

[13] M. G. Marsh, *IPROUTE2 Utility Suite Documentation.* [Online]. Available: http://www.policyrouting.org/iproute2.doc.html

[14] N. Hanford *et al.*, "Impact of the end-system and affinities on the throughput of high-speed flows," in *Proceedings of the tenth acm/ieee symposium on architectures for networking and communications systems*, 2014, pp. 259–260 [Online]. Available: http://doi.acm.org/10.1145/2658260.2661772

[15] "40G/100G tuning," *40G/100G Tuning.* ESnet [Online]. Available: https://fasterdata.es.net/host-tuning/100g-tuning/

[16] O. Maor, "Company," *Performance Tuning for Mellanox Adapters Version History.* Mellanox, Jun-2016 [Online]. Available: https://community.mellanox.com/docs/DOC-2489