Overcoming HPC System Management Challenges: An Open Source Approach

Building Tools to Solve Problems We All Have in Common

Michael Hartman
Stanford Research Computing Center (SRCC)
Stanford University
Stanford CA USA
michael.hartman@stanford.edu

Stéphane Thiell
Stanford Research Computing Center (SRCC)
Stanford University
Stanford CA USA
sthiell@stanford.edu

Kilian Cavalotti
Stanford Research Computing Center (SRCC)
Stanford University
Stanford CA USA
kilian@stanford.edu

ABSTRACT

This paper will outline four different tools developed to either solve a specific problem or streamline a workflow related to the configuration and administration of a HPC Cluster. The issues that prompted the creation of the tools were identified at the Stanford Research Computing Center in the context of managing the Sherlock HPC Cluster and Oak long-term data storage environments. The tools that were created to address the issues encountered have been used in multiple locations, both internal and external to Stanford University. In this paper, we describe the solutions developed for four different areas of system management: filesystem (Lustre), drives (SAS), interconnect (InfiniBand), and job scheduler (Slurm).

CCS CONCEPTS

• Hardware → Communication hardware, interfaces and storage → External storage • Hardware → Communication hardware, interfaces and storage → Buses and high-speed links • Networks → Network components → Intermediate nodes → Bridges and switches • Computer systems organization → Architectures → Distributed architectures → Grid computing

KEYWORDS

Lustre filesystems, Slurm configuration, InfiniBand, SAS hard drives

ACM Reference format:

Michael Hartman, Stéphane Thiell and Kilian Cavalotti. 2022. Overcoming HPC System Management Challenges: An Open Source Approach. In *Proceedings of HPC Systems Professionals Workshop 2022(HPCSYSPROS22). ACM, New York, NY, USA, 2 pages.*
https://doi.org/10.1145/1234567890

1 Lustre Filesystem Tooling: del_ost

The process for removing an Object Storage Service (OSS)/Object Storage Target (OST) from the Lustre filesystem requires several steps. The first preparatory item is changing the configuration so that no new files can be created on the OST. After the OST has been set as "no create", we begin the process of emptying the OST by either deleting the existing objects or migrating the objects off the OST. Migration is done by using either "lfs migrate" or lfs_migrate helper commands. The "lfs migrate" commands are used to re-layout the data in the existing files with the new layout parameter by copying the data from the existing OST to the new OST. lfs_migrate is a more user-friendly utility that migrates data between OSTs. Once the OST is empty, which is verified by using the "lfs find" command, we permanently disable the OST by updating the Lustre configuration logs using "lctl conf_param" command on the Lustre Management Service (MGS) to notify the Lustre Metadata Targets (MDTs) and clients of the change. Lustre was built with the assumption that the configuration logs would be append-only: that any changes would be added to the end of the file and older config records were not re-written. Therefore, it will update the configuration logs but only by appending the entries for deactivation to the end of the file. This results in the system having configuration logs that retain valid OSS/OST configuration entries at the top of the file with entries for those same systems disabled at the bottom. For normal operation this is valid until you want to add new clients: when new clients are added, they read the configuration logs from the top and process each line in a serial fashion. This causes the clients to try and connect to each OSS/OST that is still configured at the top of the file but disabled further down. Consequently, the new client mounts will have to wait for the connection attempts to time out based on a cluttered and out of date configuration file.

To clean up the configuration logs and permanently remove the old entries requires regenerating the configuration logs with the "tunefs.lustre --writeconf" command. The writeconf command should be used with caution as it is destructive to some of the configuration items such as any tunables set by conf_param and the OST pools information. The process requires unmounting the clients, MDTs and OSTs. The "tunefs.lustre –writeconf" command then needs to be run on the MDTs and the OSTs. Finally, it requires a restart of the filesystem by mounting the MDTs, the OSTs and then the clients. After that is complete, tuning configurations must be re-applied. Obviously, executing these steps requires down time on any system that Lustre is connected to; depending on the size of the Lustre system this outage can be significant. And disruptive, as users do not have access to their data for the duration of the process. Depending on the size of the filesystem and the overall workload, it may not be simple to find an opportunity to reboot the entire Lustre system.

The Stanford Oak filesystem is both large and heavily used. The 40PB system has more than 2 billion files, and it is also connected to multiple HPC clusters, each having a user base that is continually reading from and writing to the filesystem. This makes taking an outage on the system a non-trivial task: as the systems attached to the filesystem need to be drained of jobs, which can take up to two weeks (max wall time for jobs) and then the actual maintenance work on the filesystem must be performed. Overall, this creates a situation disrupting the work of hundreds of active users across multiple platforms.

We avoid this situation by using "lctl del_ost". The del_ost option leverages a specific feature in Lustre to set an EXCLUDE flag on entries in the configuration logs. As discussed earlier, entries at the top of the configuration file are typically negated by entries that have been appended to the file. The del_ost tool instead uses an EXCLUDE flag to replace the entry that is being removed. For example, if you were to remove ost3 with conf_param and dump the contents of the configuration logs you would see that there is an ost3 entry at the top of the file and an ost3 disabled entry at the bottom. If you were to make the same change with del_ost and then dump the configuration logs, it would show that for the ost3 entry there is an EXCLUDE flag. There is no longer a need to append an entry to disable ost3. Once the EXCLUDE flag has been set on the entry the MDTs and clients treat the entries as if they were not there. This has the added advantage of new clients connecting to the system also treat the entry as though it does not exist. In turn this allows new clients to bypass those lines in the configuration logs and not have to wait for the connections to time out as they normally would.

One drawback to updating the configuration logs using del_ost is that they still keep growing during the filesystem lifetime. Indeed, the configuration logs are binary files and the EXCLUDE flag "disables" the old entries in place, but this does not reduce the space used by the configuration logs, which are sent to all MDTs and all clients by the MGS. We believe this should not be an issue for Lustre unless removing and adding thousands of OSTs. Using this process, Stanford has been able to drain, remove and decommission older OSS/OSTs and add newer OSS/OST hardware with greater capacity to the Oak Lustre filesystem without needing outages. This, in turn, resulted in users' uninterrupted work, while the filesystem was changed transparently in the background.

2 sasutils

This will cover the suite of items related to sasutils. The Oak Lustre filesystem at Stanford has more than 4,500 Small Computer System Interface (SCSI) Attached Storage (known as SAS) drives total and up to 816 drives attached per server. As the filesystem grew to this size, additional tooling was needed to manage the SAS topology as the number of disks continued to scale. The sasutils suite provides several tools to help with configuration consistency as well as gather and present pertinent data: a python library with a few tools, and two helper scripts to create human readable device aliases through udev rules.

2.1 sas_counters

This tool is used to extract I/O and physical counters, as well as describe the SAS topology layout. As an option, it provides this information in a format suitable for Carbon/Graphite[1] monitoring and is compatible with SCSI Enclosure Management 2 (SES-2)[2] nicknames.

```
$ sas_counters
...
oak-io1-s1.SAS9300-
8e.0x500605b00ab01234.Switch184.io1-
sassw1.JB4602_SIM_0.io1-jbod1-
0.bays.41.ST8000NM0075.0x5000c50084c79876.ioerr_c
nt 2 1487457378
oak-io1-s1.SAS9300-
8e.0x500605b00ab01234.Switch184.io1-
sassw1.JB4602_SIM_0.io1-jbod1-
0.bays.41.ST8000NM0075.0x5000c50084c79876.iodone_
cnt 7154904 1487457378
oak-io1-s1.SAS9300-
8e.0x500605b00ab01234.Switch184.io1-
sassw1.JB4602_SIM_0.io1-jbod1-
0.bays.41.ST8000NM0075.0x5000c50084c79876.ioreque
st_cnt 7154906 1487457378
...
```

2.2 sas_discover

This tool also displays the SAS topology. It presents the topology in the tiered format and tries to fold in common devices. It also has several options to vary the level of verbosity. It's the tool you want to use to verify cabling and is especially useful when using a large SAS topology.

```
$ sas_discover -v
oak-io1-s1
|--host35 SAS9300-8e
|  `--8x--expander-35:0 ASTEK
|         |--1x--end_device-35:0:0
|         |      `--enclosure io1-sassw1 ASTEK
|         `--4x--expander-35:1 QCT
|            |-- 60 x end_device -- disk
|            `--  1 x end_device -- enclosure
io1-jbod1-0 QCT
```

```
`--host36 SAS9300-8e
   `--8x--expander-36:0 ASTEK
          |--1x--end_device-36:0:0
          |      `--enclosure io1-sassw2 ASTEK
          `--4x--expander-36:1 QCT
             |-- 60 x end_device -- disk
             `--  1 x end_device – enclosure io1-
jbod1-1 QCT
```

2.3 sas_devices

This is a zeroconf tool that is used to discover SAS devices and resolve associated enclosures. It is most often used to quickly check the hardware setup. The default output gives an overview of the enclosure with the number of associated block devices. Using the "-v" option will expand the output to show the disk devices with associated serial numbers.

An example showing a proper detection of a 60-disk JBOD (Just a Bunch Of Disks) with 2 SIMs (an "enclosure group"):

```
$ sas_devices
Found 2 SAS hosts
Found 4 SAS expanders
Found 1 enclosure groups
Enclosure group: [io1-jbod1-0][io1-jbod1-1]
NUM         VENDOR         MODEL    REV  PATHS
 60 x      SEAGATE ST8000NM0075   E002      2
Total: 60 block devices in enclosure group
```

2.4 ses_report

This provides SES status and environmental metrics. There are two options for this tool. The first is the "-c" option, which will find all the enclosures and provide metrics formatted with SES-2 nicknames and sg_ses, for consumption to by Carbon/Graphite. The second is the "-s" option, which will not provide metrics but the status of all detected SES Element Descriptors.

```
$ ses_report -c --prefix=datacenter.stanford
datacenter.stanford.io1-
sassw1.Cooling.Left_Fan.speed_rpm 19560
1476486766
datacenter.stanford.io1-
sassw1.Cooling.Right_Fan.speed_rpm 19080
1476486766
datacenter.stanford.io1-
sassw1.Cooling.Center_Fan.speed_rpm 19490
1476486766
...
```

```
# ses_report -s --prefix=datacenter.stanford |
grep SIM
datacenter.stanford.io1-jbod1-
0.Enclosure_services_controller_electronics.SIM_0
0 OK
datacenter.stanford.io1-jbod1-
0.Enclosure_services_controller_electronics.SIM_0
1 OK
datacenter.stanford.io1-jbod1-
0.SAS_expander.SAS_Expander_SIM_0 OK
datacenter.stanford.io1-jbod1-
0.SAS_expander.SAS_Expander_ISIM_2 OK
datacenter.stanford.io1-jbod1-
0.SAS_expander.SAS_Expander_ISIM_0 OK
...
```

2.5 udev scripts

The sasutils suite also includes two Zero Configuration Networking (zeroconf) udev scripts. These scripts make administering a SAS environment more manageable by creating friendly device aliases by using SES-2 sub enclosure nicknames.

2.5.1 sas_sd_snic_alias

Configuring this in your udev rules will generate udev aliases using the SES-2 sub enclosure nickname and bay identifier of each device.

For example, add the following to your udev rules:

```
KERNEL=="sd*",
PROGRAM="/usr/bin/sas_sd_snic_alias %k",
SYMLINK+="%c"
```

This should generate udev aliases made of the device sub enclosure nickname followed by the bay identifier. In the following case, *io1-jbod1-0* is the sub enclosure nickname (here SIM 0 of JBOD #1).

```
$ ls -l /dev/io1-jbod1-0-bay26
lrwxrwxrwx 1 root root 4 Oct 14 21:00 /dev/io1-
jbod1-0-bay26 -> sdab
```

2.5.2 sas_mpath_snic_alias

This utility is very similar to **sas_sd_snic_alias** but only accepts device-mapper devices. Add the following line to your udev rules:

```
KERNEL=="dm-[0-9]*",
PROGRAM="/usr/bin/sas_mpath_snic_alias %k",
SYMLINK+="mapper/%c"
```

This will result in useful symlinks.

```
$ ls -l /dev/mapper/io1-jbod1-bay26
lrwxrwxrwx 1 root root 8 Oct 14 21:00
/dev/mapper/io1-jbod1-bay26 -> ../dm-31
```

Note:
For **sas_mpath_snic_alias** to work with a JBOD having two SIMs, both enclosure nicknames should have a common prefix (eg. "myjbodX-") that will be automatically used.

2.6 sasutils python library

The library currently has four working functions.

- Listing SAS Hosts (controllers)
- Listing Expanders
- Listing Unique Expanders
- SCSI Enclosure

2.6.1 Listing SAS Hosts (controllers)

The following example will list all SAS hosts (controllers) found in sysfs.

```
from sasutils.sas import SASHost
from sasutils.sysfs import sysfs

# sysfs is a helper to walk through sysfs (/sys)
for node in sysfs.node('class').node('sas_host'):

    # Instantiate SASHost with the sas_host sysfs
device class
    host = SASHost(node.node('device'))

    # To get its sysfs name, use:
    print(host.name)
    # To get attributes from scsi_host, use:
    print('  %s' %
host.scsi_host.attrs.host_sas_address)
    print('  %s' %
host.scsi_host.attrs.version_fw)
```

Result (in this example, we have two SAS HBAs):

```
host21
  0x500123400ab06e40
  12.00.00.00
host22
  0x500123400ab09310
  03.00.08.00
```

2.6.2 Listing Expanders

The following example will list all SAS expanders found in sysfs (one instance for each path).

```
from sasutils.sas import SASExpander
from sasutils.sysfs import sysfs

# sysfs is a helper to walk through sysfs (/sys)
for node in
sysfs.node('class').node('sas_expander'):

    # Instantiate SASExpander with the
sas_expander sysfs device class
    expander = SASExpander(node.node('device'))

    # To get its sysfs name, use:
    print(expander.name)
    # To access its attributes, use:
    print('  %s' % expander.attrs.product_id)
    # To get attributes from the sas_device sysfs
class, use:
    print('  %s' %
expander.sas_device.attrs.sas_address)
```

Result:

```
expander-21:0
  Switch184
  0x50012be001234bff
expander-21:1
  JB4602 SIM 0
  0x5001636001234d7f
expander-22:0
  Switch184
  0x50012be001234c7f
expander-22:1
  JB4602 SIM 1
  0x5001636001234e3f
```

2.6.3 Listing Unique Expanders

Expanders may be accessed through multiple paths and sysfs instantiates a sas_expander for each of them. The following example shows how to list unique SAS expanders.

```
from operator import attrgetter
from itertools import groupby

from sasutils.sas import SASExpander
from sasutils.sysfs import sysfs

expanders = (SASExpander(node.node('device'))
             for node in
sysfs.node('class').node('sas_expander'))

# Find unique expander thanks to their
sas_address
attrname = 'sas_device.attrs.sas_address'
# Sort the expander list before using groupby()
expanders = sorted(expanders,
key=attrgetter(attrname))
# Group expanders by SAS address for addr,
expgroup in groupby(expanders,
attrgetter(attrname)):
    print('SAS expander %s (paths=%d)' % (addr,
len(list(expgroup))))
```

Result:

```
SAS expander 0x50012be012343bff (paths=2)
SAS expander 0x50012be012343c7f (paths=2)
SAS expander 0x5001636012342e3f (paths=2)
```

2.6.4 SCSI Enclosure

SAS is obviously very close to the SCSI layer of Linux. sasutils has a scsi module that supports SCSI types like Enclosure SCSI devices (type 13). The following example will quickly list all enclosure devices found.

```
from sasutils.scsi import EnclosureDevice
from sasutils.sysfs import sysfs

# Iterate over sysfs SCSI enclosures
for node in
sysfs.node('class').node('enclosure'):
    # Get enclosure device
    enclosure =
EnclosureDevice(node.node('device'))
    # Get enclosure SG device
    sg_dev = enclosure.scsi_generic
    print('SCSI Enclosure %s' % sg_dev.name)
    print('  %s' % enclosure.attrs.vendor)
    print('  %s' % enclosure.attrs.sas_address)
```

Result:

```
SCSI Enclosure sg124
  QCT
  0x500163600112343d
SCSI Enclosure sg1
```

```
  ASTEK
  0x50012be001234bfd
SCSI Enclosure sg63
  ASTEK
  0x50012be00012347d
SCSI Enclosure sg62
  QCT
  0x500163600123457d
```

3 Slurm GPU SPANK Tool

This will cover the information related to the gpu_cmode plugin for Slurm. It leverages the Slurm Plug-in Architecture for Node and job (K)control (SPANK)[4] to add functionality to the Slurm Scheduler. The Slurm SPANK GPU Compute Mode plugin is written in Lua[3] and require the slurm-spank-lua plugin to work. Currently there are three modes of operation for GPUs:

- Default (shared): Multiple host threads can use the device at the same time.
- Exclusive-process: Only one CUDA context may be created on the device across all processes in the system.
- Prohibited: No CUDA context can be created on the device.

For optimal performance it is recommended that the GPU compute mode be set to "Exclusive-process". This also ensures process isolation and avoids multiple processes unintentionally running on the same GPU.

There are, however, legacy applications that require processes to run concurrently on the same GPU to be able to function properly or at all. This would require the GPU compute mode to be set to "Default".

The Slurm SPANK GPU Compute Mode plugin allows users to be able to set the GPU compute mode based on their need. Since, for most applications, the optimal mode is "Exclusive-process" that is the default setting for the plugin. Using the plugin users are able to change the setting on the GPU using the "--gpu_cmode=< shared | exclusive | prohibited >" option based on their needs.

Examples of usage:

Requesting `Default` compute mode:

```
--gpu_cmode=shared

$ srun --gres gpu:1 --gpu_cmode=shared "nvidia-
smi --query-gpu=compute_mode --
format=csv,noheader"

Default
```

Requesting `Exclusive-process` compute mode:

```
--gpu_cmode=exclusive

$ srun --gres gpu:1 --gpu_cmode=exclusive nvidia-
smi --query-gpu=compute_mode --
format=csv,noheader

Exclusive_Process
```

Requesting `Prohibited` compute mode:

```
--gpu_cmode=prohibited

$ srun --gres gpu:1 --gpu_cmode=prohibited
nvidia-smi --query-gpu=compute_mode --
format=csv,noheader

Prohibited
```

4 ibswinfo

This section will cover the ibswinfo tool. This particular tool grew out of the need to remotely interact with a large number of unmanaged InfiniBand (IB) switches that were installed in the Sherlock cluster. Unmanaged IB switches are a cost-effective way to expand the fabric of a cluster, however, the downside is that they are just that: unmanaged.

NVIDIA (formerly Mellanox) provides a suite of tools called Mellanox Firmware Tools (MFT). These tools are most often used to update, or "burn", firmware updates to IB cards that are installed in servers, but they also have functionality that will allow for in-band communication to unmanaged switches and the ability to gather information related to the hardware configuration and status.

The script can gather inventory or monitoring information such as part number, serial number, Parameter-Set Identification (PSID), Global Unique Identifier (GUID), firmware version, uptime, power supply information (status, consumption, part and serial number), temperatures (including Quad Small Form Factor Pluggable (QSFP) module temp.), and fan speed/status. These items provide a significant amount of data that allows for better troubleshooting in the event of a hardware failure on an unmanaged switch.

Example output:

```
# ./ibswinfo.sh -d /dev/mst/<device>
===========================================
<node description>
===========================================
part number        | MQM8790-HS2F
serial number      | <redacted>
product name       | Jaguar Unmng IB 200
revision           | AC
ports              | 40
PSID               | MT_0000000063
GUID               | <redacted>
firmware version   | 27.2000.1886
-------------------------------------------
uptime (d-h:m:s)   | 196d-07:05:40
-------------------------------------------
PSU0 status        | OK
     P/N           | MTEF-PSF-AC-C
     S/N            | <redacted>
     DC power       | OK
     fan status     | OK
     power (W)      | 64
PSU1 status         | OK
     P/N            | MTEF-PSF-AC-C
     S/N            | <redacted>
     DC power       | OK
     fan status     | OK
     power (W)      | 47
---------------------------------------------
temperature (C)     | 34
max temp (C)        | 41
---------------------------------------------
fan status          | OK
fan#1 (rpm)         | 5426
fan#2 (rpm)         | 4746
fan#3 (rpm)         | 5426
fan#4 (rpm)         | 4798
fan#5 (rpm)         | 5426
fan#6 (rpm)         | 4815
fan#7 (rpm)         | 5382
fan#8 (rpm)         | 4868
fan#9 (rpm)         | 5471
---------------------------------------------
```

5 Conclusion

Creating new and better tools should be driven by making things easier for those that use the environment and for those who administer it. Although the four tools that we covered are very different, each was developed to do the same basic thing - make the HPC Cluster environment easier to use and manage. Tools like del_ost and the sasutils suite help manage large Lustre filesystems by making it simpler to migrate and update the systems, as well as keep configurations uniform, readable and formatted for monitoring tools. The ibswinfo tool allows for more control and better observability of Infiniband fabrics, using streamlined, unmanaged switches that can lower the operating costs compared to their managed counterparts. The Spank plugin for Slurm allows users to set the mode of the GPU environment, providing better control for users, and a path for backward compatibility within the GPU environment for legacy software. Tools such as these are increasingly essential as HPC Clusters grow and become more complex. We hope that by sharing these tools with the community, we can help other sites reduce the amount of effort needed to maintain their HPC Clusters, and we look forward to the opportunity to review tools from other sites for possible integration into our workflow.

APPENDIX

Links to the tools can be found here:

Sasutils:
https://github.com/stanford-rc/sasutils
ibswinfo:
https://github.com/stanford-rc/ibswinfo
Slurm SPANK GPU Compute Mode Plugin:
https://github.com/stanford-rc/slurm-spank-gpu_cmode
del_ost:
https://build.whamcloud.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.xhtml#lustremaint.remove_ost
https://jira.whamcloud.com/browse/LU-7668
https://review.whamcloud.com/#/c/41449/
SPANK plugin:
https://github.com/SchedMD/slurm/blob/master/slurm/spank.h

REFERRENCES
[1] Graphite Monitoring
https://graphiteapp.org
[2] SES-2
https://en.wikipedia.org/wiki/SES-2_Enclosure_Management
[3] Lua
https://www.lua.org
[4] Slurm SPANK
https://slurm.schedmd.com/spank.html