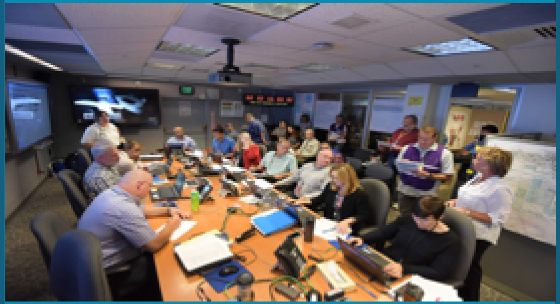
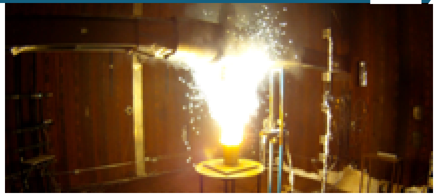
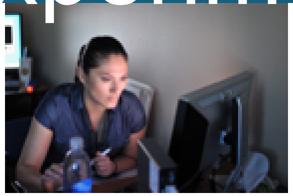
Sandia National Laboratories

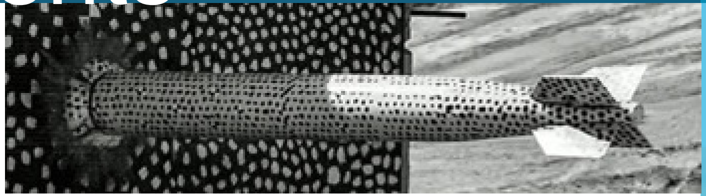
SAND2021-0714C

BeeGFS on Demand on Stria Initial Integration and Experiments









Michael Aguilar, Phil Regier, Lee Ward, Matthew L. Curry, Kevin Pedretti, Jeff Ogden

U.S. DEPARTMENT OF ENERGY
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Table of Contents

Why use an On Demand Parallel File System and why BeeGFS?

Slurm Requirements and Scalability Goals

Current Set-Up Initialization on Stria

First Attempt at Integration into Slurm on Stria

Second Attempt at Integration into Slurm on Stria

Future Work

Current Caveats

Questions?

Why use an On Demand Parallel File System and why BeeGFS?

An Ephemeral On Demand file system creates a scratch space that automatically is created for users when a job allocation is created.

Once an allocation is revoked the scratch space is cleared and removed.

Data Locality--The nodes provided in a job allocation provide the parallel object management, storage space and meta space.

Security

- Expected re-shuffling of meta and storage spaces for each Slurm allocation.

Future HPC architectures could possibly make good use of On Demand File Systems---ie. Remote Persistent Memory and 3D memory architectures.

BeeGFS already has an On Demand Parallel File System

Why use an On Demand Parallel File System and why BeeGFS?

Integrating the On Demand File System with Slurm

When a Slurm Allocation is provided, a BeeOND file system is created.

When a Slurm Deallocation is performed, the BeeOND file system is torn down.

All the compute nodes are clients of the parallel storage and simultaneously contribute to the parallel storage

Once an allocation is revoked the scratch space is cleared and removed.

Using both RAM disks and on-board SSD drives for ’proof–of-concept’.

Slurm Requirements and Scalability Goals

BeeOND Filesystem Creation Start-up Should be as Parallel as Possible

Speed of Start-Up < 20s.   Longer start-up times can cause Slurm to misbehave

Utilize Slurm Components to Avoid Redundant Actions

Fail-Over to Slurm when a File Allocation/Breakdown Issue is Encountered

For our use, MGMT and META Servers Separate from Storage, except when 1 node is Slurm Allocated

Current Set-Up Initialization on Stria

Munge is a credential authentication system.

Slurm is a job Workload and Resource Manager.

From the Beegfs.io website: "BeeGFS transparently spreads user data across multiple servers. By increasing the number of servers and disks in the system, you can simply **scale performance and capacity** of the file system to the level that you need, seamlessly from small clusters up to enterprise-class systems with thousands of nodes.”

BeeOND (BeeOND) is an ephemeral file system that we targeted to work with Slurm and Munge to build and take down a parallel scratch file system using storage on compute nodes that have been allocated for job runs.

Current Set-Up Initialization on Stria

Our methodology:

Create SLURM Prolog and Epilog scripts to build and break down the storage when SLURM allocates and deallocates nodes for users.

Modify our universal SLURM prolog and epilog to call BeeOND setup/teardown routines.

Genders attribute(s) should control when the BeeOND capability is enabled and provided on a given compute node

Add in ability to toggle on/off BeeOND for users via sbatch/srun/salloc custom options (via the Lua interfaces in Slurm)

Configuration Management allows all of the necessary support files and device drivers to load on boot/when requested.

Current Set-Up Initialization on Stria

After a basic boot-up of a compute node, *nodeup* mounts storage spaces, prepares Slurm, and in our case, prepares BeeOND

- Testing of the potential storage space, if it is using a local or remote IO storage--UDEV
- Creation of a common Device---UDEV
- Mounting of the Device on a common mount /beeond
- Verify access to allocation members, via SSH
- Automatic compiling of beegfs device driver and load into the kernel.

Current Set-Up Initialization on Stria

Nodeup:

Is the 1TB SSD node storage available for BeeOND or is that disk space being used for something else?

- Using UDEV rules, we check the space and then allocated open read/write access

  - KERNEL=="sda1",ATTRS{partition}=="1",ATTRS{size}=="1875382960",MODE="0666",SYMLINK+="beeond_store"
  - #> udevadm test /sys/class/block/sda1
  - #> udevadm control --reload-rules && udevadm trigger
  - #> udevadm info –a –p /sys/class/block/sda1

Current Set-Up Initialization on Stria

Nodeup scripts:

1) mount a block device for BeeOND use.

```
mkdir -p /mnt/beeond && mkdir -p ssd_fs
mount -t xfs /dev/beeond_store ssd_fs || true;
```

2) Compile and load a beegfs module for our compute nodes

```
modprobe beegfs
if [ $? -ne 0 ]; then
    /etc/init.d/beegfs-client rebuild
    modprobe beegfs
fi
```

Current Set-Up Initialization on Stria

Alternately, you can create a RAM-based block device.

- modprobe brd rd_size=4096000
- mkfs.xfs /dev/ram0
- mount –t xfs /dev/ram0 /ssd_fs

Current Set-Up Initialization on Stria

- We implemented a modular SLURM prolog and epilog to integrate BeeOND into our existing Slurm allocation/preparation.
  - Created "prolog.d" and "epilog.d" directories
  - Designated a point during the universal prolog/epilog at which any scripts therein would be executed and errors (if any) preserved so that the scheduler could drain the node accordingly.
  - Wrapper using the GNU "timeout" utility to catch any failures of such modules to exit in a timely manner.
  - Any nodes participating receive the BeeOND module scripts and therefore run the candidate BeeOND startup/teardown process directly from the scheduler with error handling and reporting built in.

```
function check_and_run () { test -x "$1" && "$1"}
function bail () { tee >(logger) <<<"$1"; exit $2 }
Prolog_dir=/etc/slurm/prolog.d
if test -d "$Prolog_dir"; then
    for p in "$Prolog_dir"/*; do
        check_and_run "$p" || bail "Error $? in prolog plugin '$p'" $?
    done
```

First Attempt at Integration into Slurm on Stria

Our first integration method used the beeond script, as follows:

Simple, but not well integrated into Slurm, causing issues……….

- Allocation:
  - echo $SLURM_NODELIST | hostlist -e -d "\n" > /tmp/beeond_nodefile;
  - beeond start -n /tmp/beeond_nodefile -d /ssd_fs-c /mnt/beeond -P
- Deallocation:
  - beeond stop -n /tmp/beeond_nodefile –d –L -P

First Attempt at Integration into Slurm on Stria

Issues with our first attempt:

- Large Slurm allocations took longer than 20s and thus our Slurm allocation and deallocation steps became unstable.

First Attempt at Integration into Slurm on Stria

How to overcome the problems?

- Testing SSH access to the compute nodes is unnecessary since Slurm already has access to each allocated node through Munge.
- We would like to take advantage of the Slurm tools, Prolog and Epilog, since they inherently run in parallel.
- Checking for and managing an allocated list of nodes is a duplicated step because Slurm takes care of that for us, as well.
- Calling startup and stop actions with PDSH and SSH is a duplicate step because Slurm Prolog and Epilog can manage the start up and stop actions.

Second Attempt at Integration into Slurm on Stria

Utilize Slurm Components to avoid Redundant Actions

Fail-Over to Slurm when a File Allocation/Breakdown Issue is Encountere

MGMT and META Servers Separate from Storage, except when 1 NODE is Slurm Allocated. (separating metadata and management servers from object servers is good for analyzing system behavior)

Second Attempt at Integration into Slurm on Stria

FAIL-OVER TO SLURM WHEN A FILE ALLOCATION/BREAKDOWN ISSUE IS ENCOUNTERED

```
error()
{     echo $1 1>&2;
       logger $1;                    # log the error
cleanup 1;   }
cleanup()
{     # clean up files if necessary
       logger "goodbye from beeond prolog"
       exit $1;  }
trap cleanup SIGTERM SIGINT SIGHUP
trap "" SIGINT
```

Second Attempt at Integration into Slurm on Stria

Key Code Pieces Ripped from BeeOND:

- Start up Mgmt server with Mgmtd Store Directory, log file info, PID file, Connection Port, and as a daemon
- Start up Storage servers to point at the Mgmtd server, Storage Store Directory, log file info, PID file, Connection Port, and as a daemon
- Start up Metadata server(s) to point at the Mgmtd server, Meta Store Directory, log file info, PID file, Connection Port, and as a daemon
- Start up Helperd on all of the nodes with the connection port
- Mount the file system on the nodes

Second Attempt at Integration into Slurm on Stria

Make sure that start-up steps are properly sequenced.

Timing checks also ripped from BeeOND.

beegfs-ctl --sysMgmtdHost=<node name> --connPortShift=1000 --listtargets --nodetype=storage --state

beegfs-ctl --sysMgmtdHost=<node name> --connPortShift=1000   --listtargets --nodetype=meta --state

Second Attempt at Integration into Slurm on Stria

Sanely Stopping BeeOND in Epilog

- kill -9 `pgrep bee` ; # Stop beegfs processes and don’t worry about a clean shutdown
- rm -rf /beeond/beegfs-mgmt’;rm -rf /beeond/beegfs-storage; ….
- rm /var/run/beegfs-mgmt.pid; rm /var/run/beegfs-storage.pid,….

Second Attempt at Integration into Slurm on Stria

Load the Slurm allocation into a BASH array.
Get a count of the nodes in the allocation.

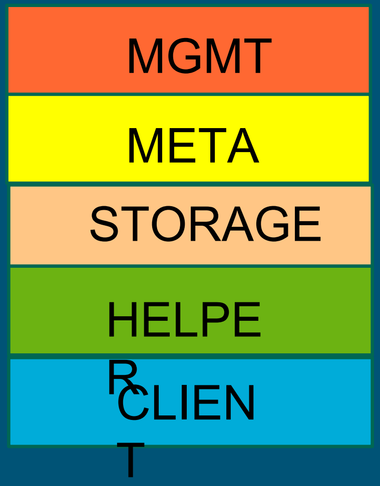
```
NODECOUNT=`echo $SLURM_NODELIST | hostlist  -e -d "\n"|wc -l` ;logger "The Slurm allocated node count provided to the BeeOND allocation is: "$NODECOUNT
….

while read HOSTENTRY; do
        NODES+=($HOSTENTRY)
        logger "The node array value for position: "$ARRAY_INDEX" is: "${NODES[$ARRAY_INDEX]}
        ARRAY_INDEX=$((ARRAY_INDEX+1))
done < <(echo $SLURM_NODELIST | hostlist  -e -d "\n”
MASTERHOST=${NODES[0]};logger "The Master Host is: "$MASTERHOST
```

Second Attempt at Integration into Slurm on Stria

case "$NODECOUNT" in

1) # We can assume that we have just 1 node allocation. Therefore, everything in the BeeOND parallel filesystem runs on just 1 node.

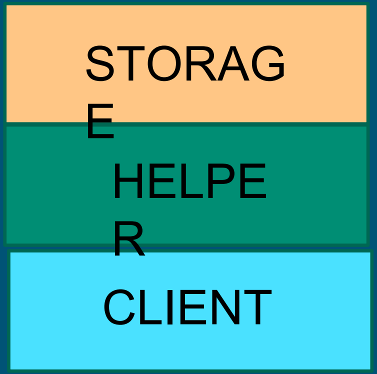


Second Attempt at Integration into Slurm on Stria

case "$NODECOUNT" in

*)   # 2 - 200 node configuration with node 1: Mgmt and Meta, nodes 2 - 199: storage

if echo "$MASTERHOST" | grep -qw "$myname"; then



Future Work

Pass parameter with RAM disk size

RAM disk in Prolog

Container support with privileged control

Support for other Schedulers and Workload Managers

Allow Slurm to access BeeOND mirroring

Support for more Metadata servers

Allowance for distribution of Metadata servers and Storage Servers

Current Caveats

Some variance in read/write performance depending upon whether a RAMdisk or SSD is used

The lowest node in the Slurm allocation always contains the Meta and Management server

You can allocate more meta servers but again they start at the bottom of the allocation

Questions?