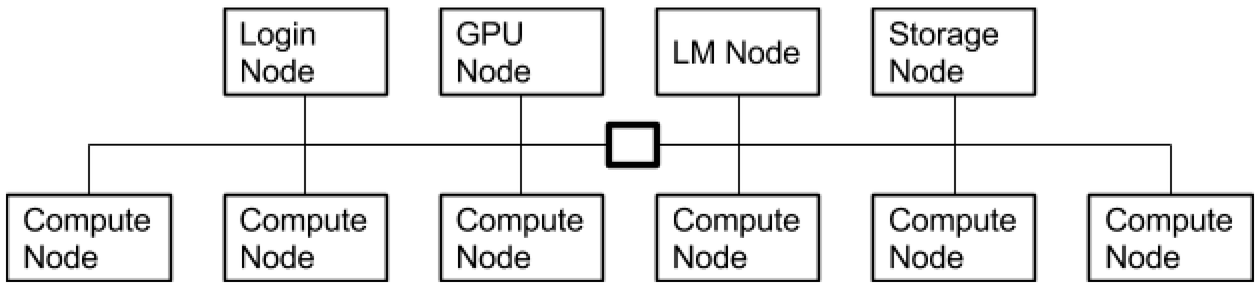
Project Assignment Part 02
Team 5-SDSC Comet
Luke Morrow, Grant Adams, Foster McLane, Thomas Rea

Diagram:



Network Topology Reasoning:
In order to replicate Comet's architecture, we attempted to keep the ratios of the node types the same in our cluster. Comet has 4 login nodes, 1944 standard compute nodes, 36 GPU nodes, 4 large memory nodes and the disk subsystem. In our cluster, we included 1 login node, 1 GPU node, 1 large memory node, 1 storage node including /home directory, and 6 standard compute nodes. We decided to run our simulation on Clemson's Cloudlab site since it has the largest memory nodes which is a key feature of Comet. We recognize that Cloudlab does not offer GPU nodes, but we wanted our simulation to reflect Comet. We accessed information on Comet from http://www.sdsc.edu/support/user_guides/comet.html

Script Explanation:
In our script, we are creating 10 nodes in Cloudlab with CentOS 7 installed and linking them together on the same LAN. Within the creation loop, we have 5 different types of nodes (login, storage, gpu, large memory, and standard compute) to create. We use if statements to identify which type of node is being created, and execute the respective startup script for the node type i.e compute.sh. All of the node scripts call from the common.sh script.

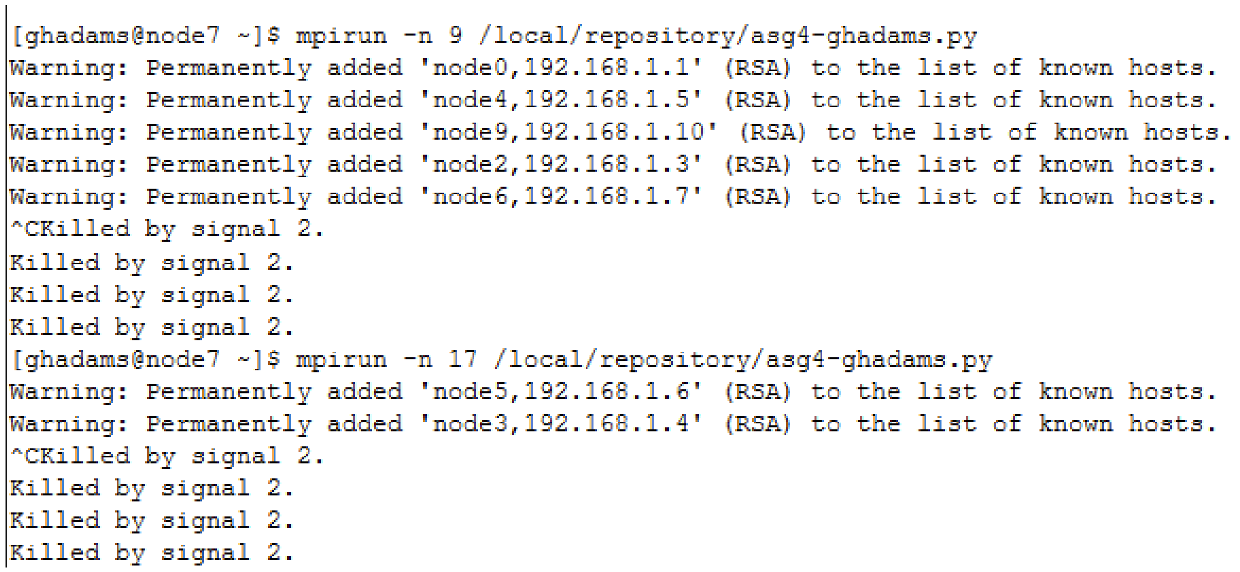
Every node first echo's its type, and then changes the shell from tcsh to bash for convenience. The script then performs node specific set-up which we will cover below. Afterwards, since every node has the same ssh key as root, we add the public key to the authorized keys for each node. We also remove strict host checking so MPI can ssh to a new host without needing user input to accept the host. This is contained in fix_ssh.

Login Node:
The login node has 1 core, 1 GB of RAM, and 4 GB of hard disk. We made this node less powerful because it only has to handle ssh sessions from users and no work is actually done on this node. The node first installs module and lustre. Then, the login node waits for the storage nodes to be ready before continuing it's setup. It ssh's into node1 to check if a file has been written, indicating that it is ready to continue. We then call setup_nfs_client and setup_lustre_client which mount the distributed directories, so that the login node can access these files.

Storage Node:
The storage node which has 1 core, 8 GB of RAM, and two block stores with 64 GB and 1024 GB for /home and /storage respectively. We install and configure the NFS server on this node next. In addition, we run a script to add keys to all the users' directories to allow for SSH'ing across all nodes. After that is completed, we install and configure lustre for this node and set up the scratch directory. The installation and configuration of lustre is very complicated and we essentially allocated blank images and mounted them to be used by every other node. This sets up mdt and ost0. Finally, we set_ready so all the other nodes can continue configuring their lustre.
GPU Node:
The GPU node is not a true GPU since it sits upon a XenVM, but it has 2 cores, 8 GB of RAM, and 4 GB of hard disk. The GPU node installs module, mpi, and lustre. After that is completed, it waits for the storage node. Once the storage node is ready, the GPU node mounts its NFS directories and sets up itself as a lustre node mounting ost2.
Large Memory Node:
The large memory node has 4 cores, 16 GB of RAM (the max allowable by CloudLab), and 4 GB of hard disk. The large memory node sets itself up identically to the GPU node just with different hardware specs.
Standard Compute Node:
These 6 nodes have 2 cores, 8 GB of RAM, and 4 GB of hard disk. The compute node sets itself up identically to the GPU node just with different hardware specs.

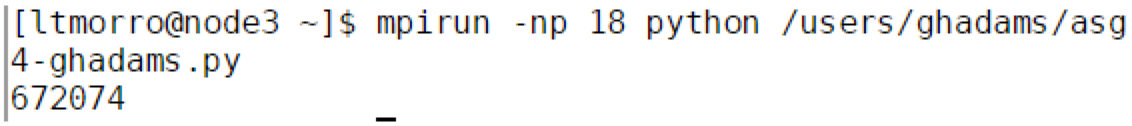
Results Validation:

To prove that our set up worked with NFS, Lustre, and MPI, we moved the job event data that we used in assignment 4 to our /scratch/ directory. We then moved the first part of assignment 4 to our home directory on NFS cluster and ran that through MPI to validate our system.



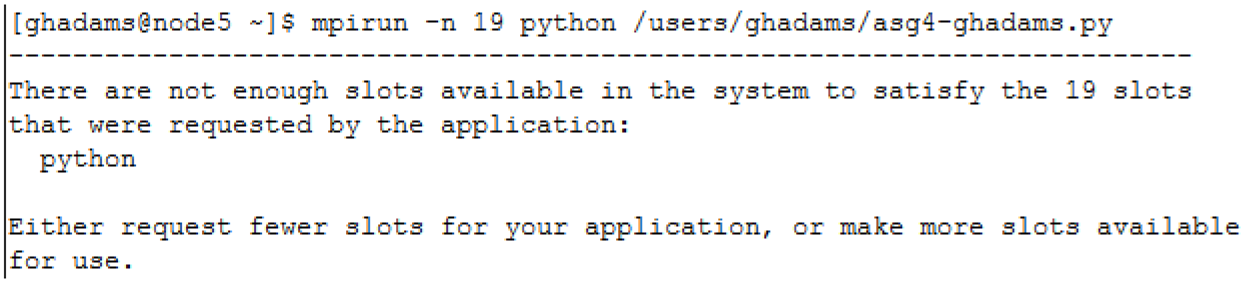
```
[ghadams@node7 ~]$ mpirun -n 9 /local/repository/asg4-ghadams.py
Warning: Permanently added 'node0,192.168.1.1' (RSA) to the list of known hosts.
Warning: Permanently added 'node4,192.168.1.5' (RSA) to the list of known hosts.
Warning: Permanently added 'node9,192.168.1.10' (RSA) to the list of known hosts.
Warning: Permanently added 'node2,192.168.1.3' (RSA) to the list of known hosts.
Warning: Permanently added 'node6,192.168.1.7' (RSA) to the list of known hosts.
^CKilled by signal 2.
Killed by signal 2.
Killed by signal 2.
Killed by signal 2.
[ghadams@node7 ~]$ mpirun -n 17 /local/repository/asg4-ghadams.py
Warning: Permanently added 'node5,192.168.1.6' (RSA) to the list of known hosts.
Warning: Permanently added 'node3,192.168.1.4' (RSA) to the list of known hosts.
^CKilled by signal 2.
Killed by signal 2.
Killed by signal 2.
Killed by signal 2.
```

In the figure above, you'll see the program being executed with 18 cores (our total number of computing cores across the cluster). It shows MPI adding each identity to the known hosts, which validates that MPI is configured correctly to execute on multiple networked nodes. After running the python script for assignment 4 against the job_events data, we got the correct number of jobs as a response.



```
[ltmorro@node3 ~]$ mpirun -np 18 python /users/ghadams/asg
4-ghadams.py
672074
```

As a proof of concept, we attempted to run mpi with 19 cores to show that it knows how many cores exist on the cluster.



```
[ghadams@node5 ~]$ mpirun -n 19 python /users/ghadams/asg4-ghadams.py
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 19 slots
that were requested by the application:
  python

Either request fewer slots for your application, or make more slots available
for use.
```