Introduction to SDSC’s Comet Supercomputer

Mahidhar Tatineni
Webinar, Feb 15, 2018
This work supported by the National Science Foundation, award ACI-1341698.
Comet
“HPC for the long tail of science”

iPhone panorama photograph of 1 of 2 server rows
Comet Built to Serve the 99%

**CHALLENGES OUR PROPOSAL ADDRESSES**
- Attract new users and communities
- Support diverse applications with complex workflows
- Ensure responsiveness for thousands of users
- Transfer, store, analyze, and share massive data sets
- Integrate with XSEDE

**COMET COMPUTE SYSTEM**

- **Cluster architecture**
  - Fast standard nodes
  - Large-memory nodes
  - GPU-accelerated nodes
  - FDR InfiniBand

- **Storage architecture**
  - Performance Storage
  - Durable Storage

- **Software**
  - Science Gateways
  - Rich base of installed apps
  - Virtualization

**USER & SYSTEM SUPPORT**
- New user orientation
- XSEDE collaborations
- FutureGrid

**ALLOCATIONS & SCHEDULING**
- Optimized for throughput
- Per-project allocation caps
- Per-job core limits

- Island architecture
- Mix of node types
- Virtualized HPC clusters

XSEDE Service Providers
- Internet2, ESnet @100G
  (Universities, Labs)
- Open Science Grid

Science Gateways

Durable Storage

Performance Storage

UCSD Campus Bridging
(e.g., LHC Tier 2 Data Site)
Comet: System Characteristics

- Total peak flops ~2.1 PF
- Dell primary integrator
  - Intel Haswell processors w/ AVX2
  - Mellanox FDR InfiniBand
- 1,944 standard compute nodes (46,656 cores)
  - Dual CPUs, each 12-core, 2.5 GHz
  - 128 GB DDR4 2133 MHz DRAM
  - 2*160GB GB SSDs (local disk)
- 72 GPU nodes
  - 36 nodes same as standard nodes plus
    Two NVIDIA K80 cards, each with dual Kepler3 GPUs
  - 36 nodes with 2 14-core Intel Broadwell CPUs plus 4 NVIDIA P100 GPUs
- 4 large-memory nodes
  - 1.5 TB DDR4 1866 MHz DRAM
  - Four Haswell processors/node
  - 64 cores/node
- Hybrid fat-tree topology
  - FDR (56 Gbps) InfiniBand
  - Rack-level (72 nodes, 1,728 cores) full bisection bandwidth
  - 4:1 oversubscription cross-rack
- Performance Storage (Aeon)
  - 7.6 PB, 200 GB/s; Lustre
  - Scratch & Persistent Storage segments
- Durable Storage (Aeon)
  - 6 PB, 100 GB/s; Lustre
  - Automatic backups of critical data
- Home directory storage
- Gateway hosting nodes
- 100 Gbps external connectivity to Internet2 & ESNet
~67 TF supercomputer in a rack

1 rack = 72 nodes
  = 1728 cores
  = 9.2 TB DRAM
  = 23 TB SSD
  = FDR InfiniBand
And 27 single-rack supercomputers

27 standard racks
  = 1944 nodes
  = 46,656 cores
  = 249 TB DRAM
  = 622 TB SSD
Comet Network Architecture
InfiniBand compute, Ethernet Storage

Node-Local Storage
72 HSWL
320 GB
320 GB
72 HSWL
36 GPU
4 Large-Memory

Management
Gateway Hosts
Login Data Mover

FDR 36p
4 switches
72

Core InfiniBand (2 x 108-port)

FDR 72
2*36

FDR 36p
72

Mid-tier InfiniBand

IB-Ethernet Bridges (4 x 18-port each)

Arista 40GbE (2x)

Data Mover

FDR 36p
4

FDR 36p
4

18 switches
72

FDR

72

Arista 40GbE (2x)

INFINIBAND

Performance Storage
7.7 PB, 200 GB/s
32 storage servers

Durable Storage
6 PB, 100 GB/s
64 storage servers

Home File Systems
VM Image Repository

Internet 2

Research and Education
Network Access
Data Movers

Additional Support Components
(not shown for clarity)
Ethernet Mgt Network (10 GbE)

7x 36-port FDR in each rack wired as full fat-tree. 4:1 over subscription between racks.

Performance Storage
7.7 PB, 200 GB/s
32 storage servers

Durable Storage
6 PB, 100 GB/s
64 storage servers

Home File Systems
VM Image Repository

Internet 2

Research and Education
Network Access
Data Movers

Additional Support Components
(not shown for clarity)
Ethernet Mgt Network (10 GbE)

7x 36-port FDR in each rack wired as full fat-tree. 4:1 over subscription between racks.
Comet Flexibility Addresses Diverse Needs

- **Wide range of hardware options**
  - Large number of regular compute nodes (1,944) with 128GB of memory and 210GB of local flash.
  - Subset of compute nodes have 1.5TB of local flash
  - 4 large memory (1.5TB RAM) nodes
  - 72 GPU nodes (36 with K80s and 36 with P100s) with 4 GPUs each.

- **Flexible Software Environment**
  - Rich set of applications (>100) in regular compute environment
  - Hadoop/Spark capability can be enabled within regular scheduler environment.
  - Supports Singularity based containerization to enable other Linux based environments (for example Ubuntu). Users can upload their own images!
  - Virtual Clusters (VC) – see operational bullet below.

- **Flexible Operations**
  - Flexible scheduler environment – shared and exclusive queues, long running jobs, focus on quick turn around time
  - Research Groups/communities, who have people in their group with expert system administration skills, can build their own virtual clusters with a custom OS and custom operational setup.
Extensive Software/Applications Stack

- Applications packaged into “Rocks Rolls” that are built and deployed on any of the SDSC systems. Benefits wider community deploying software on their Rocks clusters.

- Efficient system administration pooling software install/testing efforts from different projects/machines

- Users benefit from a familiar applications environment across SDSC systems.

- Rolls made available on Github* for all applications installed on Comet => easy replication of environment if users/groups wish to do so.

*https://github.com/sdsc/?query=roll
## Compilers, Applications, Libraries

<table>
<thead>
<tr>
<th>Category</th>
<th>List of Software/Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilers</td>
<td>Intel, PGI, GNU, MVAPICH2, OPENMPI, IntelMPI</td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>BamTools, BEAGLE, BEAST, BEAST 2, bedtools, Bismark, BLAST, BLAT, Bowtie, Bowtie 2, BWA, Cufflinks, DPPDiv, Edena, FastQC, FastTree, FASTX-Toolkit, FSA, GARLI, GATK, GMAP-GSNAP, IDBA-UD, MAFFT, MrBayes, PhyloBayes, Picard, PLINK, QIIME, RAxML, SAMtools, SOAPdenovo2, SOAPsnp, SPAdes, TopHat, Trimmomatic, Trinity, Velvet</td>
</tr>
<tr>
<td>Chemistry</td>
<td>CPMD, CP2K, GAMESS, Gaussian, MOPAC, NWChem, Q-Chem, VASP</td>
</tr>
<tr>
<td>Molecular Dynamics</td>
<td>AMBER, Gromacs, LAMMPS, NAMD</td>
</tr>
<tr>
<td>Engineering</td>
<td>ABAQUS</td>
</tr>
<tr>
<td>Data Analysis/Analytics</td>
<td>Hadoop 1, Hadoop 2 (with YARN), Spark, R, Weka, KNIME, Hadoop-RDMA, Spark-RDMA, ENVI; Singularity Enabled Apps: Torch, Keras, Tensorflow, Caffe, FEniCS</td>
</tr>
<tr>
<td>Visualization</td>
<td>VisIt, IDL; Singularity Enabled: Paraview</td>
</tr>
<tr>
<td>Numerical libraries</td>
<td>ATLAS, FFTW, GSL, LAPACK, MKL, ParMETIS, PETSc, ScaLAPACK, SPRNG, Sundials, SuperLU, Trilinios</td>
</tr>
<tr>
<td>Debugging/Profiling</td>
<td>DDT, PAPI, TAU, mpiP</td>
</tr>
<tr>
<td>GPU enabled software</td>
<td>AMBER, GROMACS, NAMD, LAMMPS, BEAST, HOOMD, mvapich2-gdr, RELION</td>
</tr>
</tbody>
</table>
Data Intensive Computing & Visualization Stack

- Comet’s high performance computing and data components enable data intensive computing
- High speed Lustre filesystem with an aggregated peak measured data rate of 100 GB/s
- Several libraries and packages enable data intensive computing and visualization:
  - R – Software environment for statistical computing and graphics
  - Weka – Tools for data analysis and predictive modeling
  - RapidMiner – Environment for machine learning, data mining, text mining, and predictive analytics
  - Matlab, Octave, IDL
  - VisIt
  - Paraview
- The myHadoop infrastructure was developed to enable use Hadoop and Spark for distributed data intensive analysis via the regular scheduler.
- Several compute and GPU based data analytics tools enabled via Singularity (more details in Singularity slide).
## Getting Allocations on Comet

<table>
<thead>
<tr>
<th>Access Mechanism</th>
<th>Allocation Limits</th>
<th>Application Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial Allocation on SDSC Comet</td>
<td>1000 SUs</td>
<td><a href="https://www.xsede.org/web/xup/allocations-overview#types-trial">https://www.xsede.org/web/xup/allocations-overview#types-trial</a></td>
</tr>
<tr>
<td>Science Gateways</td>
<td>Set by gateway (Can use your XRAC)</td>
<td><a href="http://www.sdsc.edu/services/hpc/science_gateways.html">http://www.sdsc.edu/services/hpc/science_gateways.html</a></td>
</tr>
<tr>
<td><strong>XSEDE – Start-up Allocation</strong></td>
<td>50,000 SUs 2,500 GPU SUs</td>
<td><a href="https://www.xsede.org/web/xup/allocations-overview#types-startup">https://www.xsede.org/web/xup/allocations-overview#types-startup</a></td>
</tr>
<tr>
<td>XSEDE – Research Allocation</td>
<td>10M SUs</td>
<td><a href="https://www.xsede.org/web/xup/allocations-overview#types-research">https://www.xsede.org/web/xup/allocations-overview#types-research</a></td>
</tr>
</tbody>
</table>
Access Via Science Gateways (XSEDE)

- Community-developed set of tools, applications, and data that are integrated via a portal.

- Enables researchers of particular communities to use HPC resources through portals without the complication of getting familiar with the hardware and software details. Allows them to focus on the scientific goals.

- CIPRES gateway hosted by SDSC PIs enables large scale phylogenetic reconstructions using applications such as MrBayes, Raxml, and Garli. Enabled ~320 publications in 2013 and accounts for a significant fraction of the XSEDE users.

- NSG portal hosted by SDSC PIs enables HPC jobs for neuroscientists.
Getting Started

- **System Access – Logging in**
  - Linux/Mac – Use available ssh clients.
  - ssh clients for windows – Putty, Cygwin
    - [http://www.chiark.greenend.org.uk/~sgtatham/putty/](http://www.chiark.greenend.org.uk/~sgtatham/putty/)
  - Login hosts for the SDSC Comet:
    - comet.sdsc.edu
Logging into Comet

Mac/Linux:

```
ssh username@comet.sdsc.edu
```

Windows (PuTTY):

```
comet.sdsc.edu
```
Comet: System Environment

• Modules used to manage environment for users.

• Default environment:
  
  ```
  $ module li
  
  Currently Loaded Modulefiles:
  1) intel/2013_sp1.2.144  2) mvapich2_ib/2.1  3) gnutools/2.69
  ```

• Listing available modules:
  
  ```
  $ module av
  
  /opt/modulefiles/mpi/.intel
  intelmpi/2016.3.210(default) mvapich2_ib/2.1(default)
  mvapich2_gdr/2.1(default) openmpi_ib/1.8.4(default)
  mvapich2_gdr/2.2
  
  /opt/modulefiles/applications/.intel
  atlas/3.10.2(default) lapack/3.6.0(default) scalapack/2.0.2(default)
  boost/1.55.0(default) mxml/2.9(default) slepc/3.6.2(default)
  ...
  ```
Comet: System Environment

- **Loading modules:**
  
  ```bash
  $ module load fftw/3.3.4
  $ module li
  
  Currently Loaded Modulefiles:
  1) intel/2013_sp1.2.144  3) gnutools/2.69
  2) mvapich2_ib/2.1  4) fftw/3.3.4
  ```

- **See what a module does:**
  
  ```bash
  $ module show fftw/3.3.4
  ```

  ```bash
  /opt/modulefiles/applications/.intel/fftw/3.3.4:
  module-whatis fftw
  module-whatis Version: 3.3.4
  module-whatis Description: fftw
  module-whatis Compiler: intel
  module-whatis MPI Flavors: mvapich2_ib openmpi_ib
  setenv FFTWHOME /opt/fftw/3.3.4/intel/mvapich2_ib
  prepend-path PATH /opt/fftw/3.3.4/intel/mvapich2_ib/bin
  prepend-path LD_LIBRARY_PATH /opt/fftw/3.3.4/intel/mvapich2_ib/lib
  prepend-path LIBPATH /opt/fftw/3.3.4/intel/mvapich2_ib/lib
  ```
Comet: System Environment

$ echo $PATH
/opt/fftw/3.3.4/intel/mvapich2_ib/bin:/share/apps/compute/bbftp/bin:/home/mahidhar/pdsh/bin:/opt/gnu/gcc/bin:/opt/gnu/bin:/opt/mvapich2/intel/ib/bin:/opt/intel/composer_xe_2013_sp1.2.144/bin/intel64:/opt/intel/composer_xe_2013_sp1.2.144/mpi/ib/bin/intel64:/opt/intel/composer_xe_2013_sp1.2.144/debugger/gdb/intel64_mic/bin:/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/opt/ibutils/bin:/usr/java/latest/bin:/opt/pdsh/bin:/opt/rocks/bin:/opt/rocks/sbin:/opt/sdsc/bin:/opt/sdsc/sbin:/home/mahidhar/bin

$ echo $FFTWHOME
/opt/fftw/3.3.4/intel/mvapich2_ib
Parallel Programming

• Comet supports MPI, OpenMP, and Pthreads for parallel programming. Hybrid modes are possible.

• GPU nodes support CUDA, OpenACC.

• MPI
  • Default: mvapich2_ib/2.1
  • Other options: openmpi_ib/1.8.4 (and 1.10.2), Intel MPI
  • mvapich2_gdr: GPU direct enabled version

• OpenMP: All compilers (GNU, Intel, PGI) have OpenMP flags.

• Default Intel Compiler: intel/2013_sp1.2.144; Versions 2015.2.164 and 2016.3.210 available.
Running Jobs on Comet

- **Important note:** Do not run on the login nodes - even for simple tests.
- **All runs must be via the Slurm scheduling infrastructure.**
  - Interactive Jobs: Use `srun` command:
    ```
srun --pty --nodes=1 --ntasks-per-node=24 -p debug -t 00:30:00 --wait 0 /bin/bash
    ```
  - Batch Jobs: Submit batch scripts from the login nodes. Can choose:
    - Partition (details on upcoming slide)
    - Time limit for the run (maximum of 48 hours)
    - Number of nodes, tasks per node
    - Memory requirements (if any)
    - Job name, output file location
    - Email info, configuration
# Slurm Partitions

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Max Walltime</th>
<th>Max Nodes</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute</td>
<td>48 hrs</td>
<td>72</td>
<td>Used for access to regular compute nodes</td>
</tr>
<tr>
<td>gpu</td>
<td>48 hrs</td>
<td>4</td>
<td>Used for access to the GPU nodes</td>
</tr>
<tr>
<td>gpu-shared</td>
<td>48 hrs</td>
<td>1</td>
<td>Used for shared access to a partial GPU node</td>
</tr>
<tr>
<td>shared</td>
<td>48 hrs</td>
<td>1</td>
<td>Single-node jobs using fewer than 24 cores</td>
</tr>
<tr>
<td>large-shared</td>
<td>48 hrs</td>
<td>1</td>
<td>Single-node jobs using large memory up to 1.45 TB</td>
</tr>
<tr>
<td>debug</td>
<td>30 mins</td>
<td>2</td>
<td>Used for access to debug nodes</td>
</tr>
</tbody>
</table>

- Specified using `-p` option in batch script. For example:
  
  `#SBATCH -p gpu`
**Slurm Commands**

- **Submit jobs using the `sbatch` command:**
  
  ```
  $ sbatch Localscratch-slurm.sb
  ```
  Submitted batch job 8718049

- **Check job status using the `squeue` command:**
  
  ```
  $ squeue -u $USER
  ```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8718049</td>
<td>compute</td>
<td>localscr</td>
<td>mahidhar</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Priority)</td>
</tr>
</tbody>
</table>

- **Once the job is running:**
  
  ```
  $ squeue -u $USER
  ```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8718064</td>
<td>debug</td>
<td>localscr</td>
<td>mahidhar</td>
<td>R</td>
<td>0:02</td>
<td>1</td>
<td>comet-14-01</td>
</tr>
</tbody>
</table>
Comet Compute Nodes
2-Socket (Total 24 cores) Intel Haswell Processors

Hands On Examples using:
(1) MPI
(2) OpenMP
(3) HYBRID
(4) Local scratch
Comet – Compiling/Running Jobs

• Copy and change to directory:
  
  ```
  cp -r /share/apps/examples/WEBINAR /home/$USER
  cd /home/$USER/WEBINAR/MPI
  ```

• Verify modules loaded:
  
  ```
  module list
  ```
  
  Currently Loaded Modulefiles:
  
  1) intel/2013_sp1.2.144  2) mvapich2_ib/2.1  3) gnutools/2.69

• Compile the MPI hello world code:
  
  ```
  mpif90 -o hello_mpi hello_mpi.f90
  ```

• Verify executable has been created:
  
  ```
  ls -lt hello_mpi
  ```
  
  -rwxr-xr-x 1 mahidhar sdsc 721912 Mar 25 14:53 hello_mpi

• Submit job from IBRUN directory:
  
  ```
  cd /home/$USER/WEBINAR/MPI/IBRUN
  sbatch hellompi-slurm.sb
  ```
Comet: Hello World on compute nodes

The submit script is hellompi-slurm.sb:

```bash
#!/bin/bash
#SBATCH --job-name="hellompi"
#SBATCH --output="hellompi.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#This job runs with 2 nodes, 24 cores per node for a total of 48 cores.
#ibrun in verbose mode will give binding detail

ibrun -v ../hello_mpi
```
Comet: Hello World on compute nodes

IBRUN: Command is ..~/hello_mpi
IBRUN: Command is /share/apps/examples/MPI/hello_mpi
...
...
IBRUN: MPI binding policy: compact/core for 1 threads per rank (12 cores per socket)
IBRUN: Adding MV2_CPU_BINDING_LEVEL=core to the environment
IBRUN: Adding MV2_ENABLE_AFFINITY=1 to the environment
IBRUN: Adding MV2_DEFAULT_TIME_OUT=23 to the environment
IBRUN: Adding MV2_CPU_BINDING_POLICY=bunch to the environment
...
...
IBRUN: Added 8 new environment variables to the execution environment
IBRUN: Command string is [mpirun_rsh -np 48 -hostfile /tmp/rssSvauaJA -export /share/apps/examples/MPI/hello_mpi]

node 18 : Hello world
node 13 : Hello world
node  2 : Hello world
node 10 : Hello world
Compiling OpenMP Example

- Change to the examples directory:
  ```
  cd /home/$USER/WEBINAR/OPENMP
  ```

- Compile using --openmp flag:
  ```
  ifort -o hello_openmp -openmp hello_openmp.f90
  ```

- Verify executable was created:
  ```
  [mahidhar@comet-08-11 OPENMP]$ ls -lt hello_openmp
  -rwxr-xr-x 1 mahidhar sdsc 750648 Mar 25 15:00 hello_openmp
  ```
OpenMP job script

#!/bin/bash
#SBATCH --job-name="hell_openmp"
#SBATCH --output="hello_openmp.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#SET the number of openmp threads
export OMP_NUM_THREADS=24

#Run the job using mpirun_rsh
./hello_openmp
Output from OpenMP Job

$ more hello_openmp.out

HELLO FROM THREAD NUMBER = 7
HELLO FROM THREAD NUMBER = 6
HELLO FROM THREAD NUMBER = 9
HELLO FROM THREAD NUMBER = 8
HELLO FROM THREAD NUMBER = 5
HELLO FROM THREAD NUMBER = 4
HELLO FROM THREAD NUMBER = 0
HELLO FROM THREAD NUMBER = 12
HELLO FROM THREAD NUMBER = 14
HELLO FROM THREAD NUMBER = 3
HELLO FROM THREAD NUMBER = 13
HELLO FROM THREAD NUMBER = 10
HELLO FROM THREAD NUMBER = 11
HELLO FROM THREAD NUMBER = 2
HELLO FROM THREAD NUMBER = 1
HELLO FROM THREAD NUMBER = 15
Running Hybrid (MPI + OpenMP) Jobs

• Several HPC codes use a hybrid MPI, OpenMP approach.

• “ibrun” wrapper developed to handle such hybrid use cases. Automatically senses the MPI build (mvapich2, openmpi) and binds tasks correctly.

• “ibrun -help” gives detailed usage info.

• hello_hybrid.c is a sample code, and hello_hybrid.cmd shows “ibrun” usage.
#!/bin/bash
#SBATCH --job-name="hellohybrid"
#SBATCH --output="hellohybrid.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#This job runs with 2 nodes, 24 cores per node for a total of 48 cores.

# We use 8 MPI tasks and 6 OpenMP threads per MPI task
export OMP_NUM_THREADS=6
ibrun --npernode 4 ./hello_hybrid
Hybrid Code Output

[etrain61@comet-ln3 HYBRID]$ more hellohybrid.8557716.comet-14-01.out
Hello from thread 0 out of 6 from process 2 out of 8 on comet-14-01.local
Hello from thread 3 out of 6 from process 2 out of 8 on comet-14-01.local
Hello from thread 4 out of 6 from process 2 out of 8 on comet-14-01.local
Hello from thread 5 out of 6 from process 2 out of 8 on comet-14-01.local
Hello from thread 0 out of 6 from process 3 out of 8 on comet-14-01.local
Hello from thread 2 out of 6 from process 2 out of 8 on comet-14-01.local
Hello from thread 1 out of 6 from process 3 out of 8 on comet-14-01.local
Hello from thread 2 out of 6 from process 3 out of 8 on comet-14-01.local
...
...
Hello from thread 4 out of 6 from process 7 out of 8 on comet-14-02.local
Hello from thread 2 out of 6 from process 7 out of 8 on comet-14-02.local
Hello from thread 3 out of 6 from process 7 out of 8 on comet-14-02.local
Hello from thread 5 out of 6 from process 7 out of 8 on comet-14-02.local
Hello from thread 1 out of 6 from process 6 out of 8 on comet-14-02.local
Comet: Filesystems

• **Lustre filesystems** – Good for scalable large block I/O
  • Accessible from all compute and GPU nodes.
  • `/oasis/scratch/comet` - 2.5PB, peak performance: 100GB/s. Good location for storing large scale scratch data during a job.
  • `/oasis/projects/nsf` - 2.5PB, peak performance: 100 GB/s. Long term storage.
  • *Not good for lots of small files or small block I/O.*

• **SSD filesystems**
  • `/scratch` local to each native compute node – 210GB on regular compute nodes, 285GB on GPU, large memory nodes, 1.4TB on selected compute nodes.
  • **SSD location is good for writing small files (e.g. OpenFOAM output) and temporary scratch files. Purged at the end of a job.**

• **Home directories** (`/home/$USER`)
  • Source trees, binaries, and small input files.
  • *Not good for large scale I/O.*
Using SSD Scratch

```bash
#!/bin/bash
#SBATCH --job-name="localscratch2"
#SBATCH --output="localscratch2.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#Get a list of hosts
export SLURM_NODEFILE=`generate_pbs_nodefile`
cat $SLURM_NODEFILE > nodes.list.$SLURM_JOBID
uniq nodes.list.$SLURM_JOBID > nodes.unq.list

#Change to local scratch (SSD) and run IOR benchmark
cd /scratch/$USER/$SLURM_JOBID

#Run IO benchmark
ibrun -np 48 $SLURM_SUBMIT_DIR/IOR.exe -F -t 1m -b 4m -v -v -w -k
```
Using SSD Scratch

#Change back to submit dir
cd $SLURM_SUBMIT_DIR

#List files on both nodes
for (( nn=1; nn<=$SLURM_NNODES; nn++ ))
do
  p='`sed -n $nn p nodes.unq.list`'
  echo "Files on $p"
  ssh $p /bin/ls /scratch/$USER/$SLURM_JOBID
done

#Tar back the results from each node
for (( nn=1; nn<=$SLURM_NNODES; nn++ ))
do
  p='`sed -n $nn p nodes.unq.list`'
  echo "Tar files on $p"
  ssh $p /bin/tar -cvf $SLURM_SUBMIT_DIR/node$nn.tar /scratch/$USER/$SLURM_JOBID
done
Sample output from SSD test

Using Time Stamp 1518703758 (0x6a85948e) for Data Signature
Commencing write performance test.
Thu Feb 15 06:09:18 2018

<table>
<thead>
<tr>
<th>operation</th>
<th>max (MiB)</th>
<th>min (MiB)</th>
<th>mean (MiB)</th>
<th>std dev</th>
<th>max (OPs)</th>
<th>min (OPs)</th>
<th>mean (OPs)</th>
<th>std dev</th>
<th>op grep</th>
<th>tPN</th>
<th>reps</th>
<th>fPP</th>
<th>record</th>
<th>reorderoff</th>
<th>reorderand</th>
<th>seed</th>
<th>aggsiz</th>
<th>xsize</th>
<th>blksize</th>
<th>segcnt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

write
<table>
<thead>
<tr>
<th>access</th>
<th>bw(MiB/s)</th>
<th>block(KiB)</th>
<th>xfer(KiB)</th>
<th>open(s)</th>
<th>wr/rd(s)</th>
<th>close(s)</th>
<th>total(s)</th>
<th>iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>1883.02</td>
<td>40%</td>
<td>1824.00</td>
<td>0.002134</td>
<td>0.001727</td>
<td>0.000557</td>
<td>0.001964</td>
<td>0</td>
</tr>
<tr>
<td>Operation</td>
<td>Max (MiB)</td>
<td>min (MiB)</td>
<td>Mean (MiB)</td>
<td>Std Dev</td>
<td>Max (OPs)</td>
<td>min (OPs)</td>
<td>Mean (OPs)</td>
<td>Std Dev</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
<td>---------</td>
<td>-----------</td>
<td>-----------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>write</td>
<td>1883.02</td>
<td>1883.02</td>
<td>1883.02</td>
<td>0.00</td>
<td>1883.02</td>
<td>1883.02</td>
<td>1883.02</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Max Write: 1883.02 MiB/sec (1974.49 MB/sec)

Run finished: Thu Feb 15 06:09:19 2018
Files on comet-03-48
| testFile | 0.000000  |
| testFile | 0.000001  |
| testFile | 0.000002  |
| testFile | 0.000003  |
| testFile | 0.000004  |
| testFile | 0.000005  |
| testFile | 0.000006  |
| testFile | 0.000007  |
| testFile | 0.000008  |
| testFile | 0.000009  |
| testFile | 0.000010  |
| testFile | 0.000011  |
| testFile | 0.000012  |
| testFile | 0.000013  |
| testFile | 0.000014  |
| testFile | 0.000015  |
| testFile | 0.000016  |
| testFile | 0.000017  |
| testFile | 0.000018  |
| testFile | 0.000019  |
| testFile | 0.000020  |
| testFile | 0.000021  |
| testFile | 0.000022  |
| testFile | 0.000023  |
| testFile | 0.000024  |
| testFile | 0.000025  |
| testFile | 0.000026  |
| testFile | 0.000027  |
| testFile | 0.000028  |
| testFile | 0.000029  |
| testFile | 0.000030  |
| testFile | 0.000031  |
| testFile | 0.000032  |
| testFile | 0.000033  |

Files on comet-03-54
| testFile | 0.000002  |
| testFile | 0.000007  |
| testFile | 0.000008  |
| testFile | 0.000009  |
| testFile | 0.000010  |
| testFile | 0.000011  |
| testFile | 0.000012  |
| testFile | 0.000013  |
| testFile | 0.000014  |
| testFile | 0.000015  |
| testFile | 0.000016  |
| testFile | 0.000017  |
| testFile | 0.000018  |
| testFile | 0.000019  |
| testFile | 0.000020  |
| testFile | 0.000021  |
| testFile | 0.000022  |
| testFile | 0.000023  |
| testFile | 0.000024  |
| testFile | 0.000025  |
| testFile | 0.000026  |
| testFile | 0.000027  |
| testFile | 0.000028  |
| testFile | 0.000029  |
| testFile | 0.000030  |
| testFile | 0.000031  |
| testFile | 0.000032  |
| testFile | 0.000033  |
Using SSD Scratch

- **Snapshot on the node during the run:**
  
  ```
  $ pwd
  /scratch/mahidhar/435463
  $ ls -lt
  total 22548292
  -rw-r--r-- 1 mahidhar hpss 5429526528 May 15 23:48 testFile.00000001
  -rw-r--r-- 1 mahidhar hpss 6330253312 May 15 23:48 testFile.00000003
  -rw-r--r-- 1 mahidhar hpss 5532286976 May 15 23:48 testFile.00000000
  -rw-r--r-- 1 mahidhar hpss 5794430976 May 15 23:48 testFile.00000002
  -rw-r--r-- 1 mahidhar hpss 1101 May 15 23:48 IOR_native_scratch.log
  ```

- **Performance from single node (in log file copied back):**
  
  - Max Write: 250.52 MiB/sec (262.69 MB/sec)
  - Max Read: 181.92 MiB/sec (190.76 MB/sec)
Comet GPU Nodes
2 NVIDIA K-80 Cards (4 GPUs total) per node.
Or 4 P100s per node

[1] CUDA code compile and run example
[2] Hands On Examples using Singularity to enable Tensorflow
Compiling CUDA Example

• Load the CUDA module:
  module load cuda

• Compile the code:
  cd /home/$USER/WEBINAR/CUDA
  nvcc -o matmul -l. matrixMul.cu

• Submit the job:
  sbatch cuda.sb
#!/bin/bash
#SBATCH --job-name="CUDA"
#SBATCH --output="CUDA.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH -t 01:00:00

#Load the cuda module
module load cuda

#Run the job
./matmul
Add Data Analysis to Existing Compute Infrastructure
Add Data Analysis to Existing Compute Infrastructure

Resource Manager
(Torque, SLURM, SGE)
Add Data Analysis to Existing Compute Infrastructure
Add Data Analysis to Existing Compute Infrastructure
myHadoop – 3-step Cluster

1. Set a few environment variables

   # sets HADOOP_HOME, JAVA_HOME, and PATH
   $ module load hadoop
   $ export HADOOP_CONF_DIR=$HOME/mycluster.conf

2. Run myhadoop-configure.sh to set up Hadoop

   $ myhadoop-configure.sh

3. Start cluster with Hadoop's start-all.sh

   $ start-all.sh
`#!/bin/bash
#SBATCH --job-name="Anagram"
#SBATCH --output="Anagram.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:00:00
export WRKDIR=`pwd`
myhadoop-configure.sh
start-all.sh
hadoop dfs -mkdir input
hadoop dfs -copyFromLocal $WRKDIR/SINGLE.TXT input/
hadoop jar $WRKDIR/AnagramJob.jar input/SINGLE.TXT output
hadoop dfs -copyToLocal output/part* $PBS_O_WORKDIR
stop-all.sh
myhadoop-cleanup.sh`
ANAGRAM Example

- Change to directory:
  ```bash
cd $HOME/WEBINAR/hadoop/ANAGRAM_Hadoop2
  ```

- Submit job:
  ```bash
  sbatch anagram.script
  ```

- Check configuration in directory:
  ```bash
  ls $HOME/cometcluster
  ```
Anagram Example – Sample Output

cat part-00000
...
aabcdelmnu manducable,ambulanced,
aabcdeorrsst broadcasters,rebroadcasts,
aabcdeorrst rebroadcast,broadcaster,
aabcdeorrst rebroadcast,broadcaster,
aabcedkrsw drawbacks,backwards,
aabcedkrw drawback,backward,
aabceehlnsst teachableness,cheatableness,
aabceelnnrsstu uncreatableness,untraceableness,
aabceelrrrt recreatable,retraceable,
aabceehlt cheatable,teachable,
aabceellr lacerable,clearable,
aabceelnrtu uncreatable,untraceable,
aabceelorrrstv vertebrosacral,sacrovertebral,
...
...
RDMA-Hadoop and RDMA-Spark

Network-Based Computing Lab, Ohio State University
NSF funded project in collaboration with Dr. DK Panda

- HDFS, MapReduce, and RPC over native InfiniBand and RDMA over Converged Ethernet (RoCE).

- Based on Apache distributions of Hadoop and Spark.

- Version RDMA-Apache-Hadoop-2.x 1.1.0 (based on Apache Hadoop 2.6.0) available on Comet

- Version RDMA-Spark 0.9.3 (based on Apache Spark 1.5.1) is available on Comet.

- More details on the RDMA-Hadoop and RDMA-Spark projects at:
  - [http://hibd.cse.ohio-state.edu/](http://hibd.cse.ohio-state.edu/)
RDMA-Hadoop, Spark

- Exploit performance on modern clusters with RDMA-enabled interconnects for Big Data applications.
- Hybrid design with in-memory and heterogeneous storage (HDD, SSDs, Lustre).
- Keep compliance with standard distributions from Apache.
Hands On: Anagram using HHH-M mode

#!/bin/bash
#SBATCH --job-name="rdmahadoopanagram"
#SBATCH --output="rdmahadoopanagram.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=24
#SBATCH -t 00:15:00

#Script request 3 nodes - one used for namenode, 2 for data nodes/processing

#Set modulepath and load RDMA Hadoop Module
export
MODULEPATH=/share/apps/compute/modulefiles/applications:$MODULEPATH
module load rdma-hadoop/2x-1.1.0
Hands On: Anagram using HHH-M mode

# Get the host list
export SLURM_NODEFILE=`generate_pbs_nodefile`
cat $SLURM_NODEFILE | sort -u > hosts.hadoop.list

# Use SLURM integrated configuration/startup script
hibd_install_configure_start.sh -s -n ./hosts.hadoop.list -i $SLURM_JOBID -h $HA
DOOP_HOME -j $JAVA_HOME -m hhh-m -r /dev/shm -d /scratch/$USER/$SLURM_JOBID -t /
scratch/$USER/$SLURM_JOBID/hadoop_local

# Commands to run ANAGRAM example
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -mkdir -p /user/$USER/input
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -put SINGLE.TXT /user/$USER/input/SINGLE.TXT
$HADOOP_HOME/bin/hadoop --config $HOME/conf_$SLURM_JOBID jar AnagramJob.jar /user/$USER/input/SINGLE.TXT /user/$USER/output
$HADOOP_HOME/bin/hdfs --config $HOME/conf_$SLURM_JOBID dfs -get /user/$USER/output/part* $SLURM_WORKING_DIR

# Clean up
hibd_stop_cleanup.sh -d -h $HADOOP_HOME -m hhh-m -r /dev/shm
RDMA-Hadoop: HHH-M Example

• Change to directory:
  cd $HOME/WEBINAR/hadoop/RDMA-Hadoop/RDMA-HHH-M

• Submit job:
  sbatch anagram.script
Singularity: Provides Flexibility for OS Environment

- Singularity ([http://singularity.lbl.gov](http://singularity.lbl.gov)) is a relatively new development that has become very popular on Comet.
- Singularity allows groups to easily migrate complex software stacks from their campus to Comet.
- Singularity runs in user space, and requires very little special support – in fact it actually reduces it in some cases.
- We have roughly 15 groups running this on Comet.
- Applications include: Tensorflow, Paraview, Torch, Fenics, and custom user applications.
- Docker images can be imported into Singularity.
#!/bin/bash
#SBATCH --job-name="TensorFlow"
#SBATCH --output="TensorFlow.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:k80:1
#SBATCH -t 01:00:00

#Run the job
#

module load singularity
singularity exec /share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img lsb_release -a
singularity exec /share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img python -m tensorflow.models.image.mnist.convolutional
Tensorflow via Singularity

• Change to the examples directory:
  cd /home/$USER/WEBINAR/TensorFlow

• Submit the job:
  sbatch TensorFlow.sb
Distributor ID: Ubuntu
Description: Ubuntu 16.04 LTS
Release: 16.04
Codename: xenial

I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: Tesla K80
major: 3 minor: 7 memoryClockRate (GHz) 0.8235
pciBusID 0000:85:00.0
Total memory: 11.17GiB
Free memory: 11.11GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0: Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla K80, pci bus id: 0000:85:00.0)
Extracting data/train-images-idx3-ubyte.gz

... Step 8500 (epoch 9.89), 11.6 ms
Minibatch loss: 1.601, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.9%
Test error: 0.9%
Summary

• Comet can be directly accessed using a ssh client.

• Always run via the batch scheduler – for both interactive and batch jobs. *Do not run on the login nodes.*

• Choose your filesystem wisely – Lustre parallel filesystem for large block I/O. SSD based filesystems for small block I/O, lots of small files. *Do not use home filesystem for intensive I/O of any kind.*

• Comet can handle MPI, OpenMP, Pthreads, Hybrid, CUDA, and OpenACC jobs. Singularity provides further flexibility.

• Dynamic spin up of Hadoop, Spark instances within Comet scheduler framework.