Comput Cluster Workshop
What is a compute cluster

• Bunch of individual machines tied together in a special way
• Special software is used to represent those machines as a pool of shared resources
• This software gives you ability to ask for a chunk of this pool to run your software
• Tailored to batch processing (=jobs)
  • Interactive use possible
• You don’t care on which machine your job is running
  • If you do, you can ask for specific resources to be allocated to you
History

![Graph showing the trend of CPU cores deployed, Total CINT2006, and Total CINT2006rate from 2007 to 2015.](Image)
Existing cluster

• Hardware Managed by PlatformHPC
• Jobs managed by LSF
• /scratch managed by FhGFS
Why change

• PlatformHPC is
  • Inflexible
  • Outdated
  • Vendor lock-in
  • Expensive

• Better & open solutions exist, let's use them
Our choices

- Foreman for hardware deployment
- Puppet for configuration management
- Slurm for workload management
- We kept BeeGFS for /scratch
Slurm

• “Simple Linux Utility for Resource Management”
• One of the most popular HPC schedulers
  • All new & experimental things are first developed for Slurm
• We deployed version 16.05
  • In the mean time new version already out ;}
Slurm terminology

- Step – single task run by scheduler (usually single command in job script)
- Job – resource allocation, steps run within it
- Job script – simple bash script that combines resource allocation requests and job steps
- Partition – collection of resources with some common attributes (also known as queue)
- Features – set of labels applied to different compute nodes
- Constraint – set of labels job is asking for
- Account – ID used for slurm accounting purposes (equals to primary group)
Current hardware (H1 2017)

- 8 “bigmem” Westmere nodes (240 cores)
- 84 SandyBridge nodes (1344 cores)
- 162 Haswell nodes (3888 cores)
- 2 GPU nodes:
  - gpu1 with SandyBridge and 3x K20
  - gpu2 with Broadwell and 8x P100
- BeeGFS scratch storage
State of transition (as of Q1 2017)

- 54 Haswell nodes under slurm
- 28 SandyBridge nodes under slurm
- 4 bigmem nodes under slurm
- Both gpu nodes under slurm
- Scratch still directly connected to LSF nodes
- Software environment mostly usable
  - About 20 softwares still on to-do list
Bigmem nodes

- Hardware: Dell R910
- CPU: 4x E7-4870 (10 cores, 2.4GHz, SSE4.2)
- Memory: 1TB @ 1066Mhz
- Local /tmp as tmpfs, 9TB @ 2GB/s
- Network: 10Gb/s
- Features: HT, cpu2.4GHz, net10G, westmere
- Dell end-of-support 2015/3/29, 3rd party support still available
SandyBridge nodes

- Hardware: IBM HS23
- CPU: 2x E5-2670 (8 cores, 2.6GHz, AVX)
- Memory: 256 GB @ 1600Mhz
- Local /tmp as tmpfs, 258 GB @ 3GB/s
- Network: 1Gb/s per blade, 10Gb/s per chassis
- Features: HT, cpu2.6GHz, avx, sandybridge
Haswell nodes

- Hardware: Fujitsu BX2560M1
- CPU: 2x E5-2680v3 (12 cores, 2.5GHz, AVX2)
- Memory: 256 GB @ 2133Mhz
- Local /tmp as tmpfs, 258GB @ 3GB/s
- Network: 10Gb/s per blade, 20Gb/s per chassis
- Features: noHT, HT, cpu2.5GHz, avx2, net10G, haswell
GPU nodes: gpu1

• Hardware: Supermicro
• CPU: 2x E5-2630 (6 cores, 2.3GHz, AVX)
• Memory: 64 GB @ 1333Mhz
• GPU: 3x Nvidia K20m (Kepler, 5GB memory @ 208GB/s)
• Local /tmp as tmpfs, 1.6TB @ 2GB/s
• Network: 1Gb/s
• Features: noHT, cpu2.3GHz, avx, gpu=K20, sandybridgbe
GPU nodes: gpu2

- Hardware: Supermicro
- CPU: 2x E5-2680 (28 cores, 2.4GHz, AVX2)
- Memory: 512 GB @ 2400Mhz
- GPU: 8x Nvidia P100 (Pascal, 16GB memory @ 732 GB/s)
- Local /tmp as tmpfs, 197GB @ 3GB/s
- Network: 10Gb/s
- Features: HT, cpu2.4GHz, avx2, gpu=P100, broadwell
/scratch

- Hardware: Dell + NetApp
- Memory: 256 GB per server
- Network: 40Gb/s per server
- 120 disks, 350TB usable space
- 4 NVMe cards, 16TB flash cache
Hardware lifecycle

- Gather requests throughout the year
- Shopping begins in second half of the year
- New hw deployed by end of year
- Goes into production in beginning of next year
Software environments

- Base OS: CentOS 7.3
- SEPP: /g/software/bin
  - Might still work, but no guarantees
  - Planned to be phased out
- SBgrid:
  - Commercial offering
  - source /programs/sbgrid.shrc
- Environment modules
  - module avail

Each one is designed to be the only one in use
Do not mix them or undefined things will happen
Environment Modules

- Used with Lmod
- Provided by EasyBuild
  - Repeatable software builds
  - Hardware optimized builds
    - Currently building for Nehalem, SandyBridge and Haswell
  - Large community
  - Road map towards containers
Slurm architecture
Slurm commands

- `salloc` – allocate resources and spawn a shell
- `srun` – run a single job step
- `sbatch` – submit a job script
- `scancel` – kill a running job
- `squeue` – reports the state of jobs in the queue
- `sinfo` – reports the state of queues and nodes
<table>
<thead>
<tr>
<th>User command</th>
<th>PBS</th>
<th>LSF</th>
<th>Slurm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job submission</td>
<td><code>qsub [script file]</code></td>
<td><code>bsub [script file]</code></td>
<td><code>sbatch [script file]</code></td>
</tr>
<tr>
<td>Job deletion</td>
<td><code>qdel [job id]</code></td>
<td><code>bkill [job id]</code></td>
<td><code>scancel [job id]</code></td>
</tr>
<tr>
<td>Job status (by job)</td>
<td><code>qstat [job id]</code></td>
<td><code>bjobs [job id]</code></td>
<td><code>squeue [job id]</code></td>
</tr>
<tr>
<td>Job status (by user)</td>
<td><code>qstat -u [username]</code></td>
<td><code>bjobs -u [username]</code></td>
<td><code>squeue -u [username]</code></td>
</tr>
<tr>
<td>Queue list</td>
<td><code>qstat -Q</code></td>
<td><code>bqueues</code></td>
<td><code>squeue</code></td>
</tr>
<tr>
<td>Node list</td>
<td><code>pbsnodes -l</code></td>
<td><code>bhosts</code></td>
<td><code>sinfo -N</code></td>
</tr>
</tbody>
</table>

**Rosetta stone**

- PBS: Portable Batch System
- LSF: Load Sharing Facility
- Slurm: System for Load Distribution and Resource Management
Slurm wrappers for LSF commands

- bsub
- bkill
- bjobs
- lsid
Slurm wrappers for PBS commands

- qsub
- qdel
- qstat
- qalter
- qhold
- pbsnodes
From LSF logs ...

% of jobs by requested cores

- 1 core: 80%
- 2-16 cores: 2%
- 16-48 cores: 1%
- 48-256 cores: 2%
- 256-2048 cores: 3%

% of cpu time by requested cores

- 1 core: 80%
- 2-16 cores: 10%
- 16-48 cores: 5%
- 48-256 cores: 3%
- 256-2048 cores: 2%
From LSF logs ...

% of jobs by requested memory

% of cpu time by requested memory
From LSF logs ...

% of jobs by duration

- <1min
- 1 – 5min
- 5 – 10 min
- 10min – 1h
- 1 – 10 h
- 10h – 1 day
- 1 – 30 days
- 30 – 305 days

% of cpu time by duration

- <1min
- 1 – 5min
- 5 – 10 min
- 10min – 1h
- 1 – 10 h
- 10h – 1 day
- 1 – 30 days
- 30 – 305 days
Queues

• Organized by duration
  • If you know or can estimate, tell Slum how long your job will run

• Default queue: htc
  • Max runtime 1h, max memory per core 16GB

• 1day, 1week, 1month

• Hw specific:
  • bigmem
  • gpu
Backfill scheduling
Where to find help

- Wiki: https://wiki.embl.de/cluster/
- chat.embl.org #cluster
- itsupport@embl.de
- clusterNG mailing list
- Meetings as needed
  - When there are new things to announce and explain
- Bio-IT meetings, Coding Club
For more information

• www.vi-hps.org

• www.prace-ri.eu
Exercise: login

• Use ssh to login to login.cluster.embl.de
Exercise: slurm resources

- View partitions: `sinfo -l`
- View node info: `sinfo -Nl`
- View node features: `sinfo -No "%N %f"`
Slurm node states

- Idle
- Mixed
- Allocated
- Draining
- Drained
- Down
- Unknown
Exercise: modules

- List available modules: `module avail`
- Search available modules: `module spider <modulename>`
- Detailed description of a module: `module whatis <modulename>`
- Help for a specific module: `module help <modulename>`
Exercise: toolchains

• Run `gcc -v` and observe the version
• `module load foss`
• Run `gcc -v` again and observe the version
• `module list`
• `module purge`
• `module list`
Exercise: dependencies

- module load snakemake
- module list
- module load matplotlib
- module list
- snakemake -h
- What happens?
Exercise: job environment

- module purge
- module load foss
- srun gcc -v
Exercise: interactive job

- module purge
- alloc
- hostname
- env | grep SLURM
- srun hostname
- exit
Exercise: default resources

• salloc

• srun grep Cpus_allowed_list /proc/self/status

• srun cat /sys/fs/cgroup/memory/slurm/uid_$(id -u)/job_$$SLURM_JOBID/memory.limit_in_bytes

• exit
Exercise: asking for resources

- `salloc -N 1 -n 4 --mem=500`
- `srun grep Cpus_allowed_list /proc/self/status`
- `srun cat /sys/fs/cgroup/memory/slurm/uid_$(id -u)/job_$SLURM_JOBID/memory.limit_in_bytes`
- `exit`
Exercise: asking for resources

- `salloc -N 1 -n 1 --mem=300G`
Exercise: asking for resources

- `salloc -N 1 -n 1 --mem=300G -p bigmem`

- `srun grep Cpus_allowed_list /proc/self/status`

- `srun cat /sys/fs/cgroup/memory/slurm/uid_$(id -u)/job_$SLURM_JOBID/memory.limit_in_bytes`

- `exit`
Exercise: asking for features

- `salloc -N 1 -n 4 -C HT`
- `srun grep Cpus_allowed_list /proc/self/status`
- `exit`
- `salloc -N 1 -n 4 -C noHT`
- `srun grep Cpus_allowed_list /proc/self/status`
- `exit`
## Features table

<table>
<thead>
<tr>
<th></th>
<th>avx</th>
<th>avx2</th>
<th>broadwell</th>
<th>cpu 2.3 GHz</th>
<th>cpu 2.4 GHz</th>
<th>cpu 2.5 GHz</th>
<th>cpu 2.6 GHz</th>
<th>gpu= K20</th>
<th>gpu= P100</th>
<th>haswell</th>
<th>HT</th>
<th>net10G</th>
<th>noHT</th>
<th>sand ybridge</th>
<th>west mere</th>
</tr>
</thead>
<tbody>
<tr>
<td>htc</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1day</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1week</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1month</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigmem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>gpu</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data movement

• Your work is highly data intensive
• Data and compute should be as close as possible to achieve best performance
• Slurm provides per-job $TMPDIR and $SCRATCHDIR
• Nodes have at least 250GB @ 2GB/s of TMPDIR, use it!
• If you can’t, use $SCRATCHDIR

• Use /g shares only as a source of input data and a place to store results
Example: Data movement

- This job script illustrates a method of copying input to many nodes

```bash
#!/bin/bash
#SBATCH -t 03:00
#SBATCH -p 1day
#SBATCH -N 4
#SBATCH -n 96
#SBATCH --tmp=50G

#copy to node local tmp
srun -N $SLURM_NNODES cp /g/somewhere/project/input_data $TMPDIR/

module load ...
#do stuff ...

#wrap up
srun -N $SLURM_NNODES cp $TMPDIR/results /g/somewhere/project/output
```
OpenMP

- Shared memory parallelism
- A method to parallelize within the same node
- Obeys 10+ environment variables
- Slurm sets OMP_NUM_THREADS based on cpus requested by job
Exercise: OpenMP

- Prepare this job script
- Use `sbatch` to submit it
- Vary number of tasks per node
- Observe “Number of threads” and “Best rate Triad” differences

```bash
#!/bin/bash
#SBATCH -t 00:01:00
#SBATCH -N 1
#SBATCH --ntasks-per-node 1 # vary this 1..24

module load STREAM
stream_1Kx10M
```
Exercise: OpenMP and placement

- Try --hint=compute_bound or memory_bound
- Vary number of tasks per node
- Observe “Number of threads” and “Best rate Triad” differences

```bash
#!/bin/bash
#SBATCH -t 00:01:00
#SBATCH -N 1
#SBATCH --ntasks-per-node #1..24
#SBATCH --hint=

module load STREAM
stream_1Kx10M
```
MPI

• Distributed memory parallelism
• A method to parallelize across many nodes
  • Also suitable for some problems within the same node
• Our OpenMPI build integrated with slurm
Exercise: MPI

• Submit this job script

• Observe numbers

• Add `#SBATCH -C net10G`

• Observe numbers

• Add `#SBATCH --switches=1`

• Observe numbers

```bash
#!/bin/bash
#SBATCH -t 00:05:00
#SBATCH -n 2
#SBATCH --switches=1
module load OSU-Micro-Benchmarks
echo $SLURM_NODELIST
mpirun osu_bw
mpirun osu_latency
```
Exercise: notifications

- Slurm can send you emails
- They include some job efficiency statistics
- Useful to tune your exact resource request

```bash
#!/bin/bash
#SBATCH -t 00:01:10
#SBATCH -N 1 -n 1
#SBATCH -J stress
#SBATCH --mail-type BEGIN,END,FAIL
#SBATCH --mail-user=your.mail@embl.de

do something
module load stress

cd $TMPDIR
stress -t 60 -c 1 -i 1 -m 1 -d 1
```
Exercise: GPU

• Slurm implements gpu as “generic resource” (gres)
• You can ask for some number of them
• Use constraint to select specific gpu model
• Check wiki for exact gpu/cpu hardware offers

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH -n 6
#SBATCH --mem=50G
#SBATCH -C gpu=P100
#SBATCH --gres=gpu:2

run relion on 6 cpu cores and 2 gpus
module load RELION

#do relion stuff ...```
Why is my job queued?

• Your job sits in the queue in state PENDING
• Use `scontrol show job [job id]` to understand why

```
JobId=828772  JobName=CL3d_round2K2.sh
UserId=dauden(21588)  GroupId=cmueller(574)  MCS_label=N/A
Priority=4294155964  Nice=0  Account=cmueller  QOS=normal
JobState=PENDING  Reason=Resources  Dependency=(null)
...
```
Exercise: why did my job fail?

- Submit such job script
- Use `sacct -j [jobid]` to determine exit code and failing step
- Anything non-zero is a problem
- Standard ones defined in `/usr/include/sysexits.h`
- Bash has a couple of its own
- Every software can implement its own ...

```bash
#!/bin/bash
#SBATCH -t 00:01:00
#SBATCH -N 1
#SBATCH -n 1

do something that fails ...
exit 1
```
Best practices: Slurm

• Use salloc to experiment and test
• Use srun to run single commands from your scripts or external workflow managers (such as snakemake)
• Use sbatch and job scripts for everything where you want to preserve information about environment used (module load statements)
• Use notifications to fine tune your memory and runtime requests
Best practices: R

• While capable of using multiple threads via OpenMP, no performance benefit has been seen
• Recommend to use it with -n 1
• If possible, try parallelizing it with MPI (at least three ways to do that)

• Explore alternatives (like Julia)
Best practices: GPU

- Gpu1 offers 12 cores and 3 GPUs
- Gpu2 offers 28 cores and 8 GPUs
- Slurm knows which GPU is closest to which core
- If software knows about OpenMP or MPI, try to use 2-3 cores per GPU, otherwise use 1
- Use --hint=nomultithread to tell slurm to give you cores and not threads
How to approach parallelization

• Single operation over large dataset
  • Think of splitting it into smaller chunks and do them at the same time

• If you’re doing things in loops, look for independent data
  • Typically “for [all elements of an array] do ...”

• Figure out a way to execute these loop steps in parallel
  • Use some form of shared memory model
    • Parallel loop constructs
    • Independent workers
  • Use some tool that helps you with that
One of the options: Jug

• Demo by Renato Alves
Conclusion

• To achieve best performance:
  • Put data and compute as close together as possible
  • Use memory instead of disk
  • Identify independent data and implement some parallelism on it
Q & A
Thanks