ilifu Online Training — Advanced 2 — Parallelism

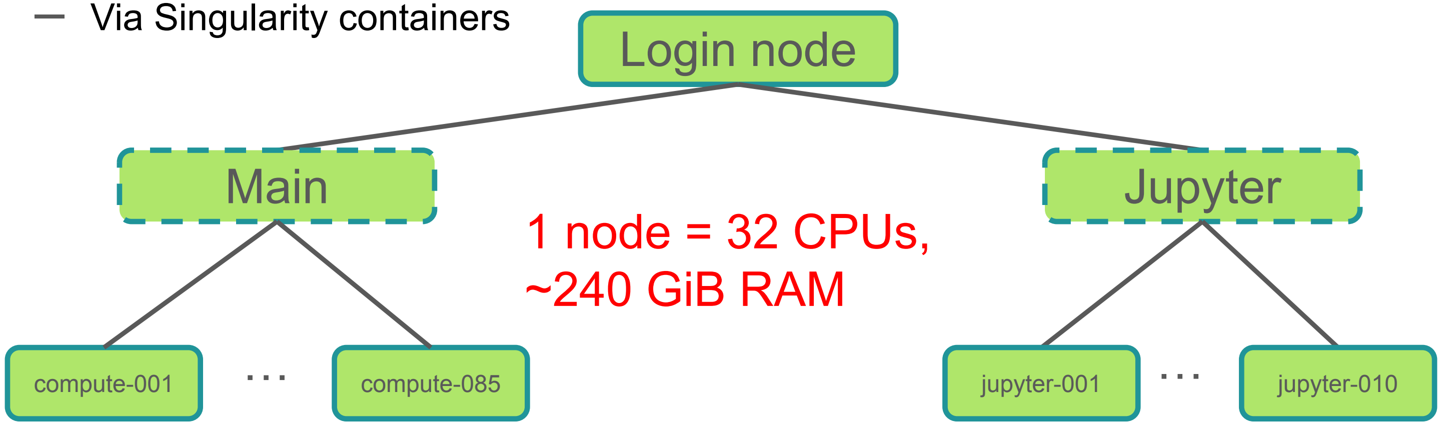
Dane Kennedy

Bioinformatics support, ilifu

April 2024

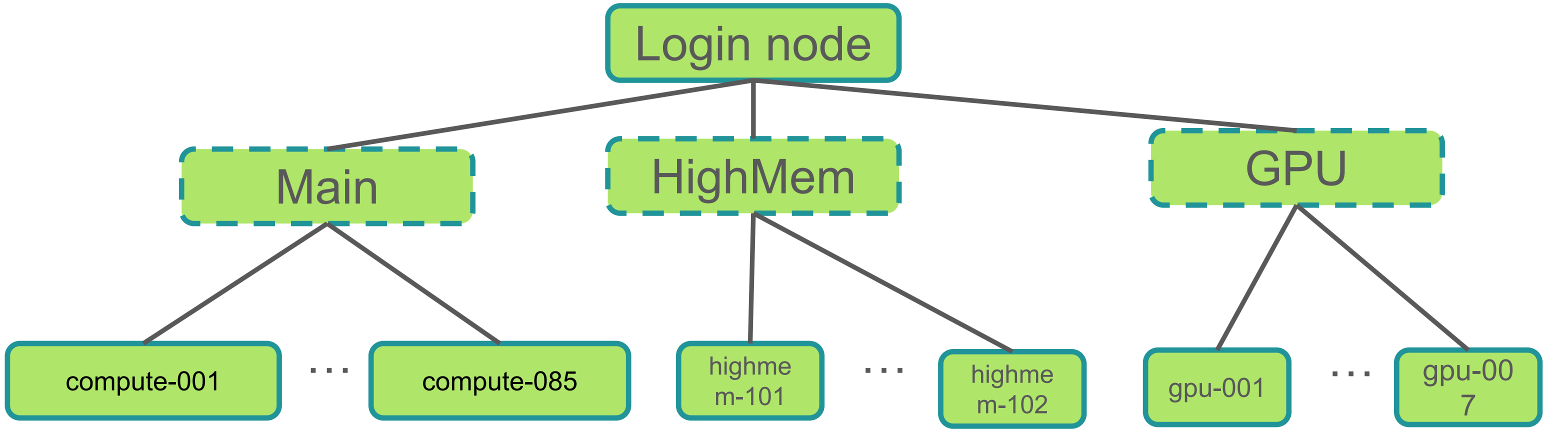
SLURM

- http://docs.ilifu.ac.za/#/getting_started/submit_job_slurm
- Login node (job submission & management)
  - Where you land when you log in (also known as “head node”)
  - **Run SLURM commands/submit jobs, but not software/heavy processes**
- Compute nodes
  - Where your processes run (also known as “worker nodes”)
  - Via Singularity containers



SLURM

- http://docs.ilifu.ac.za/#/tech_docs/running_jobs?id=_4-specifying-resources-when-running-jobs-on-slurm
- Partitions (other than Jupyter) – see with 'sinfo':
  - Main: 85 nodes (currently), each w/ 32 CPUs, 232 GB (usable) RAM
  - HighMem: 2 nodes, w/ 32 CPUs, 503 GB (usable) RAM + 96 CPU 1.5 TB RAM
  - GPU: 7 nodes (P100, V100,..), each w/ 24-48 CPUs, 232-354 GB (usable) RAM



Parallelism

- Oxford definition for parallel processing
  - *a mode of operation in which a process is split into parts, which are executed simultaneously on different processors attached to the same computer [or different computers attached to the same cluster].*
  - A cluster includes many connected nodes, each with its own RAM & CPUs
  - A node = single computer / server / VM / machine / box
- The work is partitioned into smaller jobs, sometimes with a partition of the dataset

What is a program?

- Set of discrete instructions
- Carried out sequentially
- Example: print average grade of a class

```
1.   total = 0
2.   for grade in grades:
       total = total + grade
3.   average = total / number_of_grades
4.   print(average)
```

Parallel execution of a program

- Partition grades into 2:

```
1.  total = 0
2.  for grade in 1/2 grades:

        total = total + grade

1.  average1 = total /
    number_of_grades
```

```
1.  total = 0
2.  for grade in 1/2 grades:

        total = total + grade

1.  average2 = total /
    number_of_grades
```

- Combine results

```
average = (average1 + average2) /
number_of_partitions
```

Parallelism

- Executing portions of program simultaneously
- Possible when we have many processors (cores/CPUs)
- Capacity dependent on structure of both hardware AND software
- Requires overall control/coordination mechanism
  - i.e. message passing

Parallelism on the cluster

- A cluster includes many connected nodes
- Each node has RAM and multiple cores
- Work of job is partitioned into smaller jobs
- Sometimes with a partition of the data

Parallelism

- Can be achieved on a single machine / node
  - Distributes work over many CPUs
  - Typically implemented using threads / OpenMP
- Or over multiple machines / nodes
  - Distributes work over many tasks, over 1+ nodes
  - Each given amount of memory to use
  - Generally requires a cluster
  - Typically implemented using OpenMPI
  - Requires a message passing interface (MPI) wrapper
    - mpirun, srun (SLURM), mpicasa (CASA 5)
    - Version of wrapper outside and inside container / venv must match
- Managed on ilifu by SLURM

Parallelism

- Implementing a normal job in SLURM
  - Will only use 1 CPU, 1 task, and 1 node
  - Default for many processes
- Implementing an OpenMP job in SLURM
  - Need to use >1 CPU, while nodes & tasks must be 1 (unless also using MPI)
    - `cpus-per-task` (not inherited from #SBATCH)
    - May need to export `OMP_NUM_THREADS`
- Implementing an MPI job in SLURM
  - Need to use >1 task, while nodes and CPUs can be 1
    - `nodes, ntasks-per-node, cpus-per-task`
    - Best to wrap singularity in MPI call
- Cannot exceed 32 CPUs (or tasks) per node

SLURM – serial and multi-CPU jobs

- If code is serial, i.e. doesn’t use OpenMP or MPI, increasing CPUs or nodes will not decrease execution time

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
module add openmpi
python myscript.py
```

- Using multiple CPUs within a node with OpenMP, where N is an optional number of CPUs (utilised by myscript.py)

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=N
#SBATCH --mem-per-cpu=XGB
module add openmpi
python myscript.py
```

- *Note: The maximum number of CPUs per node (32) will not always give the maximum speedup*

SLURM – multi-task and multi-node jobs

- Can also specify tasks or tasks per node

```
#SBATCH --ntasks=N
#SBATCH --cpus-per-task=1
#SBATCH --mem=XGB
module add openmpi
mpirun python myscript.py
```

- Above example doesn’t require knowledge of number of node’s CPUs; below one does

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=N
#SBATCH --cpus-per-task=1
#SBATCH --mem=XGB
module add openmpi
mpirun python myscript.py
```

SLURM – multi-task and multi-CPU jobs

- Using multiple nodes with MPI

```
#SBATCH --nodes=N
#SBATCH --ntasks-per-node=n
#SBATCH --cpus-per-task=1
#SBATCH --mem=XGB
module add openmpi
mpirun python myscript.py
```

- *Note: Need to consider that internode communication is slower than intranode communication*

- --mem is memory per node, so N times XGB allocated overall (usable by some software)

- Using multiple nodes with MPI as well as multiple cores within node with OpenMP (utilised by myscript.py)

```
#SBATCH --ntasks=N
#SBATCH --cpus-per-task=n
module add openmpi
mpirun python myscript.py
```

SLURM – dependencies

- Allows jobs to be scheduled for running, based on the status of a previous job
  - e.g. only begin a particular job once previous one successfully completes

```
$ sbatch -d afterok:882242 --kill-on-invalid-dep=yes another_job.sh
#submit another_job.sh to SLURM queue, to begin after jobID 882242
successfully completes (exit code 0), or cancel the job if jobID 882242 fails

$ sbatch -d afterany:882242:882243 another_job.sh
#submit another_job.sh to SLURM queue, to begin after jobIDs 882242 & 882243
complete (any exit code)
```

Best practices

- Don’t run software / heavy processes / scp on the login node
  - Only submit jobs and run SLURM commands (sbatch, srun, squeue, etc)
  - Use transfer.ilifu.ac.za to transfer data (external/internal), not login node
- Before running a large job, identify the available resources
  - Use sinfo. Don’t hog the cluster. Reduce your allocation if possible
  - Increase likelihood of jobs running with less memory and less walltime
- Use sbatch (srun / screen / tmux / mosh are volatile)
- Cleanup files that aren’t needed
  - Old raw data, temporary products, /scratch data, etc
- Don’t place large files in your home directory (/users)
- Use Singularity (you cannot install software on the nodes)

THANK YOU

Acknowledgements

Dr Jordan Collier for the slides

Jeremy, Tinus and Mike for all your help

IDIA Inter-University Institute for Data Intensive Astronomy

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

CBIO Computational Biology @ UCT