HPC the Easy Way

Tools and techniques for making the most of your resources

RSE Sheffield Seminar Series
University of Sheffield
30 July 2019

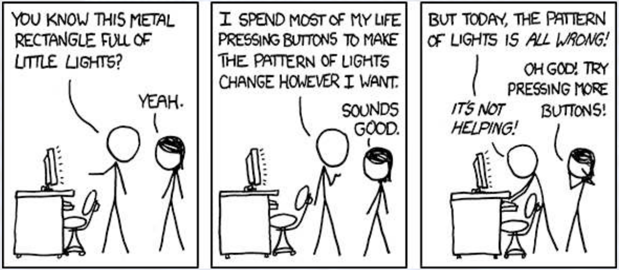
Phil Tooley
HPC Application Analyst

Outline

Common HPC Problems

Two common HPC problems

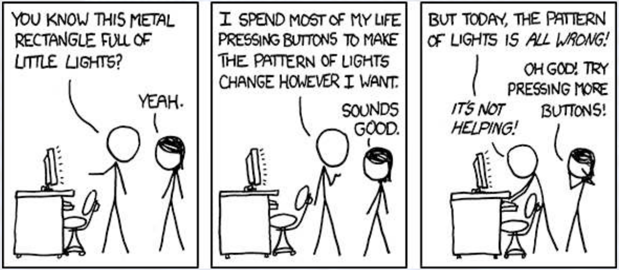
- Why is my job still queuing?
- How do I install <package>?



Common HPC Problems

Two common HPC problems

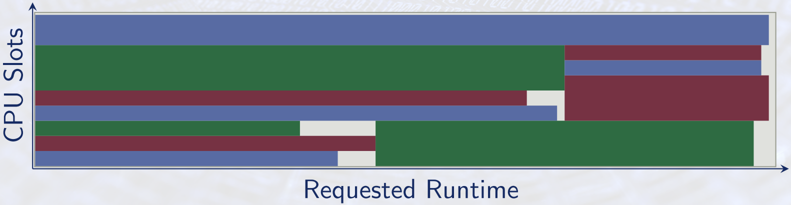
- Why is my job still queuing?
- How do I install `<package>`?



What the Scheduler does

A bin-packing problem

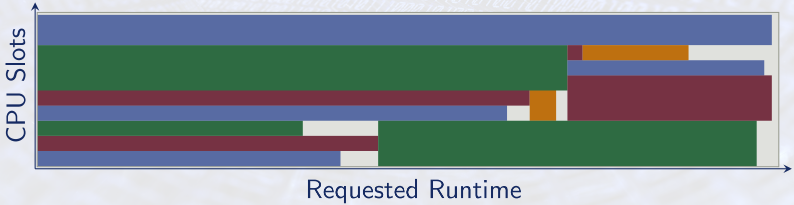
- Plans how to map jobs into nodes as efficiently as possible
- No job should wait "too long"
- Everyone should get a "fair share"
- Small jobs fill gaps around big ones



What the Scheduler does

A bin-packing problem

- Gaps appear as jobs finish early or are cancelled
- Scheduler backfills gaps as best it can
- Smaller jobs have more chances to backfill
- Ask for only what you actually need



The real world picture - ShARC

Mining the scheduler data

- Who is using ShARC?
- How are they using it?
- How efficiently are they using it?

The real world picture - ShARC

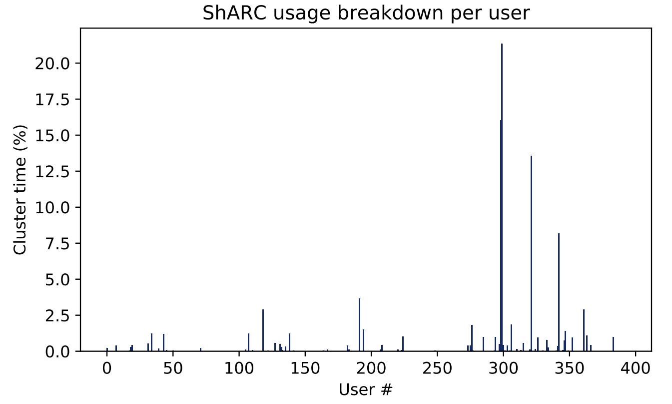
Mining the scheduler data

- Who is using ShARC?
- How are they using it?
- How efficiently are they using it?

The dataset

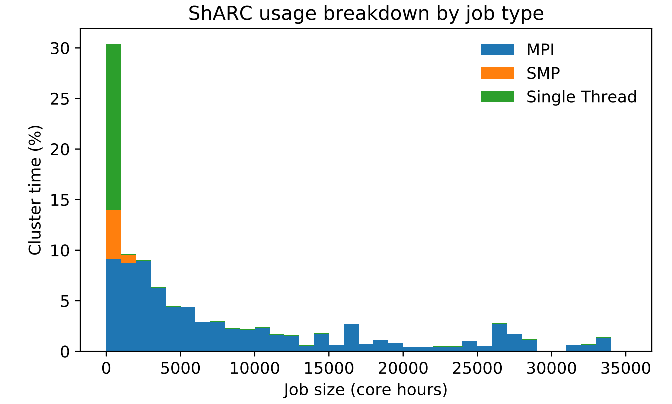
- Jobs started between 1/7/2017 – 30/6/2018
- Only public node data
- Failed jobs removed
- Sysadmin test jobs removed

User Breakdown



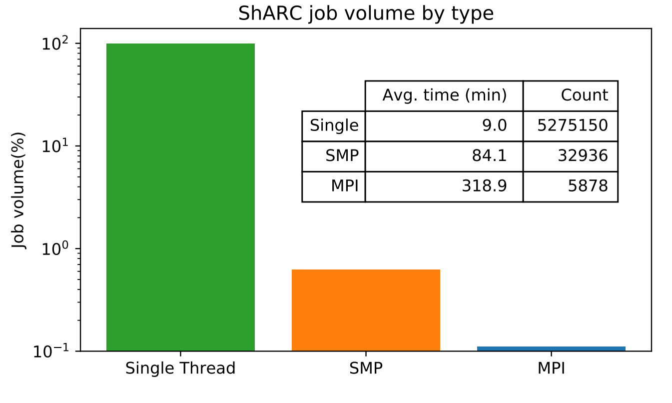
- 539 unique users
- Heaviest 3 users consumed over 50% of available cpu time

Job breakdown



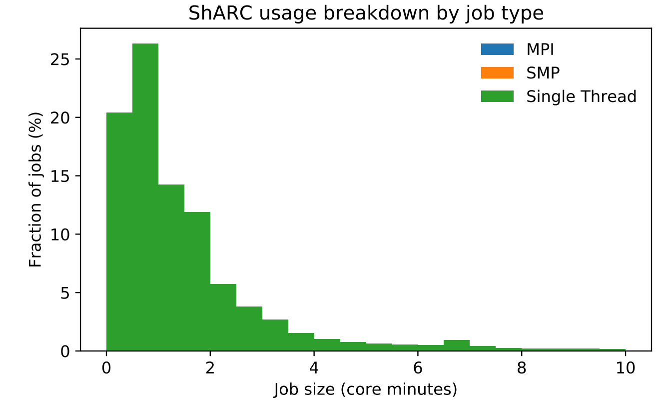
- Most time is spent running MPI jobs
- $\sim 75\%$ MPI vs. $\sim 25\%$ single node/thread

Jobs breakdown



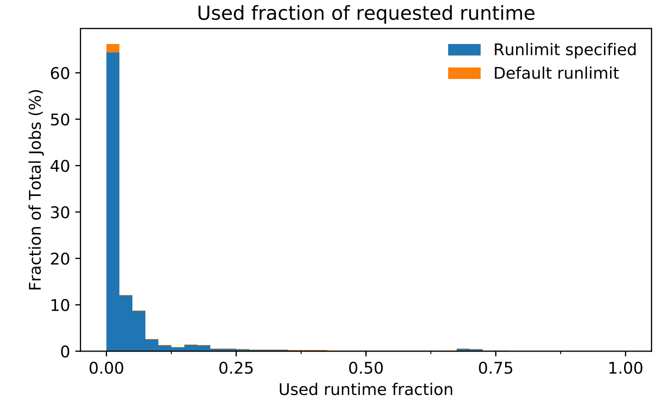
- Huge volume of very short jobs
- Heaviest users submitting $> 10^6$ short jobs each!

Jobs breakdown



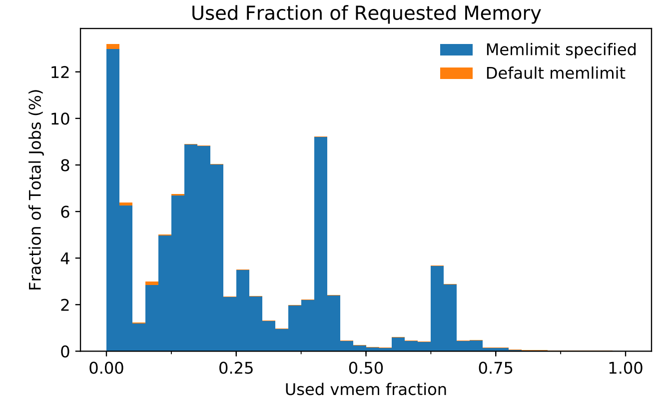
- $\sim 50\%$ of ShARC jobs shorter than 1 minute
- 50% of scheduler effort spent on only 0.4% of cpu time!

Runtime Requests and Usage



- Most over-request walltime by at least an order of magnitude
- $\rightarrow$ Lots of missed opportunities to backfill gaps!

Memory Requests and Usage



- Majority of users explicitly request memory
- Better usage, but still lots of over-requesting

Getting Feedback from the Scheduler

Accounting Information

- ShARC/Iceberg

```
$ qacct -j $jobid
```

- Bessemer

```
$ sacct -j $jobid
```

- Records basic performance information about job
  - Requested resources (time, memory etc.)
  - Actual runtime
  - Actual memory usage
  - Useful CPU time

Accounting Information

qacct -j 1150879

```
qname             all.q
hostname          sharc-node147.shef.ac.uk
owner             ac1mpt
job_number        1150879
submission_time   2018-04-16 10:00:43
start_time        2018-04-16 10:00:54
end_time          2018-04-19 10:34:48
exit_status       0
ru_wallclock      261234
granted_pe        mpi
slots             220
cpu               57314572.128644
category          -u ac1mpt -l h_rt=345600,h_vmem=2G
                  -pe mpi 220 -P SHEFFIELD
maxvmem           150.63G
```

Resource Rules of Thumb

Runtime

- Check `ru_wallclock` — actual run time
- Request 1.5–2× `ru_wallclock`

Resource Rules of Thumb

Runtime

- Check `ru_wallclock` — actual run time
- Request 1.5–2$\times$ `ru_wallclock`

Memory

- Check `maxvmem` — peak job memory usage
- Request 1.5–2$\times$ `maxvmem`
- Remember requests are *per core*

Resource Rules of Thumb

Runtime

- Check `ru_wallclock` — actual run time
- Request 1.5–2× `ru_wallclock`

Memory

- Check `maxvmem` — peak job memory usage
- Request 1.5–2× `maxvmem`
- Remember requests are *per core*

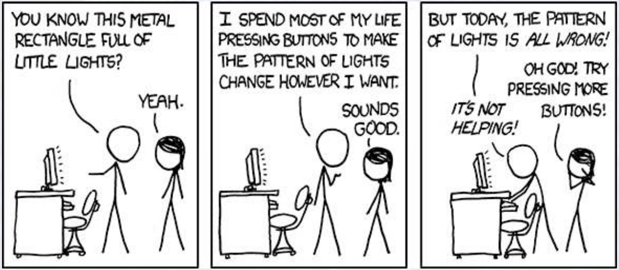
Efficiency

- Check `cpu` — actual cpu usage
- Ensure `cpu` ≃ `ru_wallclock` × `slots`

Common HPC Problems

Two common HPC problems

- Why is my job still queuing?
- How do I install <package>?



Automating Software Installation

Package Managers

- Automate installation/removal of software
- Manage installation of required dependencies
- Curate package repositories
- Document and reproduce environments

Focus on just two:

CONDA Spack

Conda

Pre-built packages for Python, R, etc.

- Originally for Anaconda Python distribution
- Microsoft provided R packages
- Low level numerical support libraries
- Intel Python with MKL optimised Numpy/Scipy
- Designed for users to install what they need

Installing Conda

Personal machine — Windows, Mac, Linux

- Two versions:
- Anaconda — Full distribution with hundreds of packages
- Miniconda — Just Conda and Python
- Download from anaconda.com and run installer

Installing Conda

Personal machine — Windows, Mac, Linux

- Two versions:
- Anaconda — Full distribution with hundreds of packages
- Miniconda — Just Conda and Python
- Download from anaconda.com and run installer

ShARC, Bessemer, Iceberg

- Already installed:

```
$ module load conda
```

Installing and Managing Packages

Conda Environments

- Collections of packages and their dependencies
- Isolate individual projects
- Test/use multiple versions of a package
- Easily capture and reproduce environment elsewhere

Installing and Managing Packages

Conda Environments

- Collections of packages and their dependencies
- Isolate individual projects
- Test/use multiple versions of a package
- Easily capture and reproduce environment elsewhere

Creating Environments

```
$ conda create --name myenv numpy pystan
$ source activate myenv
```

Installing and Managing Packages

Lots of customization options

- Choose Python version:

```
$ conda create --name myenv numpy pystan python=3.7
```

Installing and Managing Packages

Lots of customization options

- Choose Python version:

```
$ conda create --name myenv numpy pystan python=3.7
```

- Package versions:

```
$ conda create --name myenv numpy pystan=2.17.1
```

Installing and Managing Packages

Lots of customization options

- Choose Python version:

```
$ conda create --name myenv numpy pystan python=3.7
```

- Package versions:

```
$ conda create --name myenv numpy pystan=2.17.1
```

- Other channels, e.g Intel Python

```
$ conda create --channel intel --name myenv numpy
```

Installing and Managing Packages

Lots of customization options

- Choose Python version:

```
$ conda create --name myenv numpy pystan python=3.7
```

- Package versions:

```
$ conda create --name myenv numpy pystan=2.17.1
```

- Other channels, e.g Intel Python

```
$ conda create --channel intel --name myenv numpy
```

- Non Python environments e.g R:

```
$ conda create --channel r --name myRenv r rstudio
```

Using Environments

Activating and deactivating

- "Activate" an environment to use it:

```
$ conda activate myenv
```

Using Environments

Activating and deactivating

- "Activate" an environment to use it:

```
$ conda activate myenv
```

- Installed Packages are now available to use:

```
$ python
Python 3.6.8 (default, Mar 10 2019, 17:04:16)
  >>> module load pystan
  >>> module load numpy
  >>> # etc...
```

Using Environments

Activating and deactivating

- "Activate" an environment to use it:

```
$ conda activate myenv
```

- Installed Packages are now available to use:

```
$ python
Python 3.6.8 (default, Mar 10 2019, 17:04:16)
  >>> module load pystan
  >>> module load numpy
  >>> # etc...
```

- "Deactivate" the environment to exit:

```
$ conda deactivate
```

Using Environments

Installing extra packages

- Can add extra packages to the environment

```
$ conda activate myenv
$ conda install scipy scikit-learn #etc...
```

- And remove unneeded ones

```
$ conda remove scikit-learn #etc...
```

Using Environments

Installing extra packages

- Can add extra packages to the environment

```
$ conda activate myenv
$ conda install scipy scikit-learn #etc...
```

- And remove unneeded ones

```
$ conda remove scikit-learn #etc...
```

Updating packages

- Update all packages to the latest version:

```
$ conda activate myenv
$ conda update --all
```

Exporting Environments

Preserving Environments

- Export complete list of packages with versions to a file:

```
$ conda env export --name myenv > myenv.txt
```

Exporting Environments

Preserving Environments

- Export complete list of packages with versions to a file:

```
$ conda env export --name myenv > myenv.txt
```

Recreating Environments

- Now take that package list to another machine:

```
$ conda create --name myenv_clone -f myenv.txt
```

- `myenv_clone` is now an exact copy of `myenv`
  - Collaboration with other users
  - Porting to new machines
  - Publishing for reproducibility
- Plain text file listing packages — can also be created/edited by hand

Conda — Summary

Python and R Package Management

- Designed for portability and reproducibility
- Rapidly install Python, R etc. packages
- Full control of package versioning
- Maintain multiple custom package environments
- Export, share and duplicate environemnts

CONDA

Spack

Build scientific packages from source

- Primarily designed for HPC package management
- Build optimised packages for specific system
- "Recipes" to install over 3000 packages
- Interoperates with already installed packages
- For sysadmins and end-users

Installing Spack

Requirements

- Python $>= 2.6$
- A working compiler (gcc, intel, pgi, etc.)

Installing Spack

Requirements

- Python $>= 2.6$
- A working compiler (gcc, intel, pgi, etc.)

Installation

```
$ cd $HOME
$ git clone https://github.com/spack/spack.git
$ export SPACK_ROOT="$HOME/spack"
$ source $SPACK_ROOT/share/spack/setup-env.sh
```

- Install as user in homedir
- Use .bashrc to automatically set up

Configuring Spack

Compiler autodetection

```
$ spack compilers
==> Available compilers
-- gcc sles12-x86_64 ------------------------
gcc@4.8
```

Configuring Spack

Compiler autodetection

```
$ spack compilers
==> Available compilers
-- gcc sles12-x86_64 ------------------------
gcc@4.8
```

Additional compilers

```
$ module load gcc/8.1.0
$ spack compiler find
==> Added 1 new compiler:
    gcc@8.1.0
```

Configuring Spack

System packages

- Often want to use some system packages, e.g:
  - Vendor optimised MPI
  - System supplied BLAS/LAPACK
  - Avoid compiling again
- Specify in `packages.yaml`

```yaml
# /home/phil/.spack/linux/packages.yaml
packages:
  netlib-lapack:
    modules: lapack/3.8.0
    buildable: False
```

Installing Packages

Search available packages

```
$ spack list mpi
    ==> 21 packages.
    intel-mpi  mpibash  mpiblast  mpich  openmpi ...
```

Installing Packages

Search available packages

```
$ spack list mpi
    ==> 21 packages.
    intel-mpi  mpibash  mpiblast  mpich  openmpi ...
```

Install a package

- Install "preferred" version

```
$ spack install openmpi
```

- Specify a version

```
$ spack install openmpi@2.1.0
```

Spack — Summary

HPC Package Management

- A heavy duty package manager
- Designed for flexibility and control
- Integration with system modules and packages
- Full control of package versioning
- Build optimised packages from source

Spack

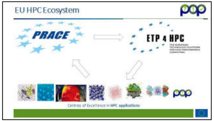
Parallel Performance Optimization and Productivity

EU H2020 Centre of Excellence (CoE)

1 December 2018 – 30 November 2021

Grant Agreement No 824080

POP CoE



- A **Centre of Excellence**
  - On **Performance Optimisation and Productivity**
  - Promoting **best practices in parallel programming**
- Providing **FREE Services**
  - Precise understanding of application and system behaviour
  - Suggestion/support on how to refactor code in the most productive way
- **Horizontal**
  - Transversal across application areas, platforms, scales
- **For (EU) academic AND industrial codes and users !**

Partners

- **Who?**
  - BSC, ES (coordinator)
  - HLRS, DE
  - IT4I, CZ
  - JSC, DE
  - NAG, UK
  - RWTH Aachen, IT Center, DE
  - TERATEC, FR
  - UVSQ, FR

A team with

- Excellence in performance tools and tuning
- Excellence in programming models and practices
- Research and development background AND
  proven commitment in application to real academic and industrial use cases

Motivation

Why?

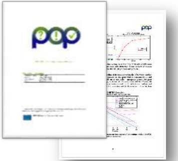
- Complexity of machines and codes
  ⇨ Frequent lack of quantified understanding of actual behaviour
  ⇨ Not clear most productive direction of code refactoring
- Important to maximize efficiency (performance, power) of compute intensive applications and productivity of the development efforts

What?

- Parallel programs, mainly MPI/OpenMP
  - Although also CUDA, OpenCL, OpenACC, Python, …

FREE Services provided by the CoE

- **Parallel Application Performance Assessment**
  - Primary service
  - Identifies performance issues of customer code (at customer site)
  - If needed, identifies the root causes of the issues found and qualifies and quantifies approaches to address them (recommendations)
  - **Combines former Performance Audit (?) and Plan (!)**
  - Medium effort (1-3 months)



- **Proof-of-Concept (✓)**
  - Follow-up service
  - Experiments and mock-up tests for customer codes
  - Kernel extraction, parallelisation, mini-apps experiments to show effect of proposed optimisations
  - Larger effort (3-6 months)

Note: Effort shared between our experts and customer!

The Process …

When?

December 2018 – November 2021

How?

- Apply
  - Fill in small questionnaire describing application and needs https://pop-coe.eu/request-service-form
  - Questions? Ask pop@bsc.es
- Selection/assignment process
- Install tools @ your production machine (local, PRACE, …)
- Interactively: Gather data → Analysis → Report