SO:UK Data Centre

Release 0.4.1

Kolen Cheung

May 17, 2024

CONTENTS

CHAPTER

ONE

ONBOARDING

You will need to

1. have a grid certificate,
2. register to a VO,
3. and obtain a UNIX account.

1.1 Grid certificate

Go to UK Certification Authority Portal and obtain a grid certificate. You will obtain a `certBundle.p12` file which is "an archive file format for storing many cryptography objects as a single file"[1]

Note: You may need to use a different website if you are outside UK. TODO: add other countries' links here.

Follow the instruction over there to obtain grid certificate. They implements a strong principle of Web of trust and requires you to physically verify your identity in-person with an existing member of the Grid.

1.2 Registering the SO:UK Virtual Organization (VO) and obtain a grid certificate

Go to VOMS Admin > souk.ac.uk and submit an application to the VO.

Note: They prefer you to use your institutional email here. It does not have to match your email address used in the Grid Certificate.

Once submitted, you will receive an email. Once you clicked the link in the email to confirm your application, you need to wait for admin to process it.

Occasionally the application went into a blackhole, feel free to ping the admin there or us if you aren't approved in a couple of days.

---

[1] See PKCS 12 - Wikipedia.

1.3 Obtain a UNIX account

Provide the following info and send it to one of us:

0. Your email address, GitHub account name, and preferred name.
1. Your preferred UNIX username. If you don't know, run `echo $USER` in your terminal and copy that. A recommended pattern is first initial then surname. For example, `jdoe`.
2. Your first name and last name, same as the one used above.
3. Copy your ssh public key(s) to us. For example, you can run the following commands and copy the first available result:

```
cat ~/.ssh/id_ed25519.pub
cat ~/.ssh/id_rsa.pub
```

> **Note:** For advanced users, if your ssh comment from your key contains your email address, you can specify an alternative comment to be used on our system. Otherwise it will be assigned by us. This is a policy enforced by Blackett.

4. (Optional) For typical users, please skip this step. If you need `sudo`, follow the following procedure to generate a salted, hashed password. *On your local machine* (`vm77`'s `openssh` is too old), enter this line,

```
$ openssl passwd -6 -salt $(openssl rand -base64 16)
# type the password you want to use in the prompt
# and result would look like this:
$6$...$...
```

   Copy the resulting string to us.

For maintainers to add this new user, go to *this section*.

CHAPTER

TWO

LOGGING INTO OUR SUBMIT NODE vm77

From the username you got from *the previous section*, here by named `$USER`, you now should be able to

```
ssh $USER@vm77.tier2.hep.manchester.ac.uk
```

and land into our submit node `vm77`.

2.1 Setting up ssh config

The recommended way to connect to our submit node would be to edit your ssh config file at `~/.ssh/config` and add these lines, (change `$USER` to your actual username here, and you can skip this line if your client machine is of the same username.)

```
Host blackett
    HostName vm77.tier2.hep.manchester.ac.uk
    User $USER

# this is for ssh into worker nodes on Blackett which loads its own temporary keys
Host condor-job.*
    IdentitiesOnly yes
    AddKeysToAgent no
```

You can then `ssh blackett` instead.

Tip: If you cannot log in at this point, first, check which key is the one you sent to us from *the previous section*. For example, if the key you sent starts with `ssh-ed25519`, then probably you are using `~/.ssh/id_ed25519.pub`. If it starts with `ssh-rsa`, then probably you are using `~/.ssh/id_rsa.pub`.

You can also list all your available keys by running this command:

```
 find ~/.ssh -name '*.pub'
```

Knowing that, you can add a line specifying `IdentityFile` to your ssh config:

```
Host blackett
    HostName vm77.tier2.hep.manchester.ac.uk
    User $USER
    IdentityFile ~/.ssh/id_ed25519
...
```

Replace `id_ed25519` with `id_rsa` if you sent us a `ssh-rsa` key instead.

If it still does not work for you, run the below command and send us the text file `ssh-debug.txt`:

```
find ~/.ssh -name '*.pub' > ssh-debug.txt
echo '===' >> ssh-debug.txt
cat ~/.ssh/config >> ssh-debug.txt
echo '===' >> ssh-debug.txt
ssh blackett -vvv >> ssh-debug.txt 2>&1
```

CHAPTER

THREE

QUICK START

As a quick start to go right into the current recommended way to run an MPI job, go through

1. The table in the beginning of the *User guide* if you are a NERSC user,
2. *Running pipelines* to have an overview,
3. *OpenMPI with CVMFS* for the current recommended way to start an MPI job,
4. Learn how to read from and write to the grid storage system
    - by setting up the *User credentials*,
    - *Mounting the grid storage system as a POSIX filesystem* on `vm77` to read from and write to it,
    - *Accessing the grid storage system from worker nodes*.

Optionally, go to *Interactive job* for starting interactive job.

CHAPTER

FOUR

RUNNING PIPELINES

4.1 Writing ClassAd

4.1.1 Vanilla universe

To request a job in the vanilla universe, create a file `example.ini`,

```
executable     = /bin/echo
arguments      = "hello world"
output         = hello_world.out
error          = hello_world.err
log            = hello_world.log
stream_output  = True
stream_error   = True
queue
```

And then submit your job using

```
condor_submit example.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit example.ini; tail -F hello_world.log hello_world.out hello_world.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

Note: You'd see that output files are automatically transferred back to your submit node. We will soon see how to specify manually what to transfer, which is especially important if you have some output files under some directories.

Explanation

`executable = /bin/echo`
Purpose: Defines the program or script to be executed.

Explanation: The job will run the `echo` command, which is typically located in the `/bin/` directory.

`arguments = "hello world"`
Purpose: Provides the arguments to be passed to the `executable`.

Explanation: The `echo` command will be invoked with the argument "hello world", causing it to print this string.

`output = hello_world.out`
Purpose: Specifies the file to capture the standard output of the job.

Explanation: Any output produced by the `echo` command (in this case, "hello world") will be written to a file named `hello_world.out` in the job's submission directory.

`error = hello_world.err`
Purpose: Specifies the file to capture the standard error of the job.

Explanation: If the `echo` command produces any error messages, they will be written to this file. For this simple example, it's unlikely there will be any errors.

`log = hello_world.log`
Purpose: Designates the file where HTCondor will write log messages related to the job's execution.

Explanation: This file will contain logs about the job's lifecycle, such as when it was started, if it was evicted, and when it completed.

`stream_output = True`
Purpose: Determines if the standard output should be streamed to the output file in real-time.

Explanation: By setting this to `True`, the `hello_world.out` file will be updated in real-time as the job produces output. This can be useful for monitoring long-running jobs.

`stream_error = True`
Purpose: Determines if the standard error should be streamed to the error file in real-time.

Explanation: Similar to `stream_output`, this ensures that the `hello_world.err` file is updated in real-time if the job produces any error messages.

`queue`
Purpose: This command tells HTCondor to add the job to its queue.

Explanation: Once this ClassAd is submitted using the `condor_submit` command, HTCondor will schedule the job to run on a suitable machine in its pool.

4.1.2 Parallel universe

To request a job in the parallel universe, create a file `example.ini`,

```
universe       = parallel
machine_count  = 2
request_cpus   = 2
request_memory = 1024M
request_disk   = 10240K
executable     = /bin/echo
arguments      = "hello world from process $(Node)"
output         = hello_world-$(Node).out
error          = hello_world-$(Node).err
```

(continues on next page)

(continued from previous page)

```
log            = hello_world.log
stream_output  = True
stream_error   = True
queue
```

And then submit your job using

```
condor_submit example.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit example.ini; tail -F hello_world.log hello_world-0.out hello_world-0.
↪err hello_world-1.out hello_world-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

Explanation

universe = parallel
: This specifies that the job you're submitting is a parallel job. In HTCondor, the `universe` attribute defines the type of environment or execution context for the job. In the case of the `parallel` universe, it allows for the coordination of multiple job processes that will run simultaneously.

machine_count = 2
: This indicates that the job requires two machines (or slots) from the HTCondor pool. Essentially, the job is requesting two instances of itself to run concurrently.

request_cpus = 2
: This asks for two CPUs for each instance (or slot) of the job. So, for the two machines specified by `machine_count`, each machine should have at least 2 CPUs.

request_memory = 1024M
: This is a request for each machine (or slot) to have at least 1024 Megabytes (1 Gigabyte) of memory.

request_disk = 10240K
: This requests that each machine (or slot) has at least 10240 Kilobytes (10 Megabytes) of available disk space.

executable = /bin/echo
: This specifies the executable that will be run. In this case, it's the `echo` command commonly found on UNIX-like systems.

arguments = "hello world from process $(Node)"
: Here, the `arguments` attribute specifies what arguments will be passed to the `echo` command. The `$(Node)` is a placeholder that gets replaced with the node (or process) number when the job runs. So, for a parallel job running two instances, you'd see one instance printing "hello world from process 0" and the other "hello world from process 1".

output = hello_world-$(Node).out
: This specifies where the standard output of each job process should be written. Using the `$(Node)` placeholder, each process will write its output to a unique file. For instance, "hello_world-0.out" for the first process, "hello_world-1.out" for the second, and so on.

error = hello_world-$(Node).err
: Similarly, this defines where the standard error of each job process should be written. For instance, any errors from the first process would go to "hello_world-0.err", from the second to "hello_world-1.err", and so on.

log = hello_world.log
: This is a consolidated log file for the job. It will contain logging information from all instances of the job, such as when each instance starts, stops, etc.

stream_output = True
: This means that the standard output of the job will be streamed (written in real-time) to the specified output file, rather than being buffered and written at the end of the job.

stream_error = True
: Similarly, this streams the standard error of the job to the specified error file in real-time.

queue
: This final command actually submits the job (or jobs, if more than one) to the HTCondor scheduler. It tells HTCondor that the job is ready to be matched with available resources in the pool.

4.1.3 Transferring files

Implicit transfer of `executable`

The simplest example that involve file transfer is the job script itself.

create a file `repl.ini`,

```
universe                = parallel
executable              = repl.sh

log                     = repl.log
output                  = repl-$(Node).out
error                   = repl-$(Node).err
stream_error            = True
stream_output           = True

should_transfer_files   = Yes
when_to_transfer_output = ON_EXIT

machine_count           = 2
request_cpus            = 2

request_memory          = 512M
request_disk            = 1G

queue
```

and a file `repl.sh`,

```
#!/bin/bash -l

# helpers #####################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
```

(continues on next page)

(continued from previous page)

```
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

########################################################################

print_double_line
echo "hostname: $(hostname)"
print_line
echo "CPU:"
print_line
lscpu
echo Hello from $_CONDOR_PROCNO of $_CONDOR_NPROCS

print_double_line
echo "HTCondor config summary:"
print_line
condor_config_val -summary

print_double_line
echo "Current environment:"
print_line
env | sort

print_double_line
echo "Avaiable MPI:"
module avail mpi

module load mpi/openmpi3-x86_64

print_double_line
echo "Current environment:"
print_line
env | sort

print_double_line
echo "module path:"
which mpicc
which mpirun
```

This ClassAd involve transferring a script named `repl.sh`, and be default it will be copied to worker nodes.

And then you can submit your job using

```
condor_submit repl.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit repl.ini; tail -F repl.log repl-0.out repl-0.err repl-1.out repl-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

Note: We normally won't use the `module` system here, but if needed, notice the shebang `#!/bin/bash -l` is

necessary for `module` to be found.

This example also includes some information specific to HTCondor that you can play around.

Explicit file transfer

Create a file `cat.ini`,

```
universe                = parallel
executable              = /usr/bin/cat
arguments               = cat.txt

log                     = cat.log
output                  = cat-$(Node).out
error                   = cat-$(Node).err
stream_error            = True
stream_output           = True

transfer_input_files    = cat.txt
should_transfer_files   = Yes
when_to_transfer_output = ON_EXIT

machine_count           = 2
request_cpus            = 2

request_memory          = 512M
request_disk            = 1G

queue
```

Over here, we use `transfer_input_files` to specify which input files to be copied to the worker nodes. If it is a relative path, it will be the relative path w.r.t. the current directory that you are submitting the job from.

To prepare the file for `transfer_input_files`, let's create `cat.txt` with the content,

```
hello world
```

And then submit your job using

```
condor_submit cat.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit cat.ini; tail -F cat.log cat-0.out cat-0.err cat-1.out cat-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

If you want to transfer more than one files, delimit them with a comma, like so:

```
transfer_input_files = file1,file2,/path/to/file3
```

In this section we will see a few typical scenario that we want to configure our job.

Examples used in this section can also be found under `1-classad/` in this repository relative to this file.

4.2 Software deployment

4.2.1 The tarball method

We will start by explaining the concept of `PREFIX` directory, and then provide an example utilizing this concept and the machinery around the conda ecosystem. We will also provide a starter tarball that you can feel free to use and modified from.

`PREFIX` directory and software deployment

`PREFIX` directory refers to a directory within which a piece of software is installed into. Usually, there are a few directories inside this prefix among others, such as `bin`, `lib`, `share`. For example, `bash` is typically installed under the prefix `/usr`.

The complication which arises in software deployment from one computer (such as submit node or your local machine) to another computer is that the followings must match:

1. OS, say Linux,
2. CPU architecture, such as x86-64 (also known as x64, x86_64, AMD64, and Intel 64),
3. prefix directory,
4. any other dependencies outside this prefix directory.

We will not talk about cross-compilation here, so we assume (1) & (2) matches already. E.g. you have a local machine running x86-64 Linux and deploying software to our data centre.

The reason prefix directory has to match is that not all software is designed to be portable. Unless you are sure the software you installed or the application you are developing is designed to be portable, you should assume it is not.

(4) can also creates complications, as if no extra care is given, your softwares inside the prefix directory will most likely depends on other things, such as dynamically linked libraries, outside the prefix directory. In this situation, if you archive your prefix directory to a tarball and ship it to a worker node, it can complains about missing dynamically linked libraries for example.

To solve (3) & (4), we utilizes the heavy machinery from conda which enables one to create a conda environment to any prefix directory of your choice, and bundle all the necessary dependencies within that prefix directory such that this prefix directory is now *self-contained*.

The last piece of the puzzle is to choose a prefix directory that both your local machine and all the worker nodes has write access to. Here we abuse the `/tmp` directory as this always exists and writable.

Note: `/tmp` are local to an HTCondor process, so even if your job might be sharing the same physical node with another job, they have their own `/tmp` directory and therefore it would not be interfered.

The provided tarball below is doing exactly this. And if you want to deploy your own applications, be sure to follow these advices to avoid any potential problems.

Example

create a file `tarball.ini`,

```
executable              = tarball.sh

log                     = tarball.log
output                  = tarball.out
error                   = tarball.err
stream_error            = True
stream_output           = True

should_transfer_files   = Yes
transfer_input_files    = /opt/simonsobservatory/pmpm-20230718-Linux-x86_64-OpenMPI.
↪tar.gz
when_to_transfer_output = ON_EXIT

request_cpus            = 16
request_memory          = 32999
request_disk            = 32G

queue
```

This ClassAd uses `transfer_input_files` to transfer a tarball from the submit node to the worker node.

Warning: The path `/opt/simonsobservatory/pmpm-20230718-Linux-x86_64-OpenMPI.tar.gz` is provided by us and may be changed over time. Try `ls /opt/simonsobservatory` to see available tarballs. File an issue if what you're seeing in the documentation is outdated.

The ClassAd involve a script `tarball.sh`,

```
#!/bin/bash -l

# helpers #############################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

#######################################################################

print_double_line
echo "Unpacking environment..."
tar -xzf pmpm-20230718-Linux-x86_64-OpenMPI.tar.gz -C /tmp
. /tmp/pmpm-20230718/bin/activate /tmp/pmpm-20230718
print_line
echo "Environment is available at:"
which python
```

(continues on next page)

(continued from previous page)

```
which mpirun

print_double_line
echo "Running TOAST tests..."
python -c "import toast.tests; toast.tests.run()"
```

Here you see that the tarball is unarchived to `/tmp`, and then the conda environment is activated explicitly using `. "$PREFIX/bin/activate" "$PREFIX"` (`PREFIX=/tmp/pmpm-20230718` in this specific example).

And then you can submit your job using

```
condor_submit tarball.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit tarball.ini; tail -F tarball.log tarball.out tarball.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

Note: We would note that in this specific example, it takes around 50s to unarchive the environment. So for long running jobs this becomes negligible. But if you starts multiple short running jobs, this is not an optimal method to deploy software.

"Forking" the provided tarball

You can modifies our provided tarball to your liking by following the above example and unarchive it under `/tmp` and activate it `. "$PREFIX/bin/activate" "$PREFIX"`. Here you can start to install more packages by using either `mamba install ...`, `conda install ...` or `pip install ...`.

Once your environment is ready,

```
# PREFIX=pmpm-20230718 in this example
cd /tmp; tar -cf "$PREFIX-Linux-x86_64-NAME.tar" "$PREFIX"
gzip "$PREFIX-Linux-x86_64-NAME.tar"
```

Now you can transfer this tarball to the submit node first (say via `rsync` or `wget`/`curl`), and then specify that in your ClassAd with `transfer_input_files`.

> **Warning:** Do not do this on the submit node. Firstly because submit node is for submission only and is very resource constrained, and also because in the submit node everyone is sharing the same `/tmp` directory. A recommended to do this is to first request an interactive node and works from there. Alternatively, do it in a dedicated Linux machine.

4.2.2 CVMFS

According to CernVM-FS documentation,

> CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace `/cvmfs`.

The key here is that it is read-only, suitable for software deployment. While there is a caching mechanism, it means that it has high latency on first launch on a node that has never fetch this software before.

We will see how it works in the following example.

Example

create a file `cvmfs.ini`,

```
executable              = cvmfs.sh

log                     = cvmfs.log
output                  = cvmfs.out
error                   = cvmfs.err
stream_error            = True
stream_output           = True

when_to_transfer_output = ON_EXIT

request_cpus            = 16
request_memory          = 32999
request_disk            = 32G

queue
```

The ClassAd involve a script `cvmfs.sh`,

```
#!/bin/bash -l

# helpers #####################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

###############################################################################

CVMFS='/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/opt/miniforge3'

print_double_line
echo "Loading a conda environment from CVMFS: $CVMFS"
. "$CVMFS/bin/activate"
```

(continues on next page)

(continued from previous page)

```
print_line
echo "Environment is available at:"
which python
```

Here, we see that `CVMFS='/cvmfs/...'` is defined. The example path given here is a minimal conda environment.

In this script, the environment is first activated, and then it shows you the environment is loaded successfully, as evident by seeing which Python it is loading.

As usual, you can submit the job via

```
condor_submit cvmfs.ini
```

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit cvmfs.ini; tail -F cvmfs.log cvmfs.out cvmfs.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

The output of this job will be

```
========================================================================
Loading a conda environment from CVMFS: /cvmfs/northgrid.gridpp.ac.uk/
↪simonsobservatory/opt/miniforge3
------------------------------------------------------------------------
Environment is available at:
/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/opt/miniforge3/bin/python
```

General softwares available from CVMFS

For your convenience, some commonly used system softwares are included in `/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/usr/bin`.

Feel free to include it in your `PATH` such as by

```
export PATH="/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/usr/bin:$PATH"
```

This includes `curl`, `ffmpeg`, `git`, `htop`, `nano`, `ranger`, `tar`, `tmux`, `tree`, `zsh`, `zstd`.

For a full list, run `ls /cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/usr/bin`.

The list of softwares may change in the future. Their existence is not guaranteed. Please let us know if you have requests on system softwares that you'd like to be supported.

4.2.3 Modifying our maintained software environments for software development

When you are using the provided software environments provided by us, you may still want to add more packages that you use, perhaps one that you are developing, that are not currently maintained by us. This includes the case that it might be a new change you are making that is not merged in a released version (such as a GitHub Pull Requests you're working on).

In these cases, if this package is something you can install via pip, this section provide a method to install it inside your job.

We will see how it works in the following example.

Example

create a file `pip-install-user.ini`,

```
universe       = vanilla
executable     = pip-install-user.sh
output         = pip-install-user.out
error          = pip-install-user.err
log            = pip-install-user.log
stream_output  = True
stream_error   = True

request_cpus   = 1
request_memory = 1G
request_disk   = 1G
queue
```

The ClassAd involve a script `pip-install-user.sh`,

```
#!/bin/bash -l

# helpers #################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

#########################################################################

print_double_line
CONDA_PREFIX=/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/conda/so-conda-py310-
↪20240104
. "${CONDA_PREFIX}/bin/activate"
echo "Conda environment loaded with python available at:"
which python
export PATH="/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/usr/bin:$PATH"

print_double_line
echo "Note that this package doesn't exist yet:"
python -c 'import souk; print(souk.__file__)'

print_double_line
echo 'Installing souk from pip:'
pip install --user souk

print_double_line
echo 'Note that this package now exists:'
python -c 'import souk; print(souk.__file__)'
which souk_arch_info
```

This example uses a package `souk` that was not in the original environment to demonstrate how this method works.

This uses `pip install --user ...` to install a package locally without having write access to our provided environment.

Note that this method includes not only a package listed in PyPI, but also a GitHub branch, such as

```
pip install --user https://github.com/simonsobs-uk/data-centre/archive/master.zip
```

As usual, you can submit the job via

```
condor_submit pip-install-user.ini
```

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit pip-install-user.ini; tail -F pip-install-user.log pip-install-user.out␣
↪pip-install-user.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

4.2.4 Continuous Deployment (CD)

Note: Continuous deployment (CD) is actively being developed. It may changes in the future.

Currently, CD is done via GitHub Actions in this repository: so-software-environment.

You can find the latest releases in https://github.com/ickc/so-software-environment/releases/latest.

Explanations of different environments

Note: This is for advanced users only. Feel free to jump to the *Examples* directly for the recommended generic choice.

`PREFIX`

`cvmfs_northgrid.gridpp.ac.uk_simonsobservatory`: for deploying to CVMFS by Content Managers, see *CVMFS* and *OpenMPI with CVMFS* for examples.

`tmp`: for deploying to `/tmp` directory, see *The tarball method* for examples.

Environment type

`so-conda`: a (pure) conda environment as defined in so-software-environment/examples/so.yml

`so-pmpm`: a conda environment with some packages compiled and optimized for specific architectures, utilizing pmpm—Python manual package manager. Currently, only `libmadam` and `toast3` are compiled here.

Python version

The `py310` string means Python 3.10. In principle we can deploy to 3.8, 3.9, 3.10 where some SO packages are currently still incompatible with 3.11, 3.12. We will only support Python 3.10 for now, and look into supporting newer versions of Python as packages become ready.

MKL

`mkl`: uses Intel MKL.

`nomkl`: uses openblas instead. AMD CPUs should uses these environments. MKL runs slower on AMD CPUs unless some form of hack is used to disguise AMD CPUs into Intel's.

x86-64-v?
: Microarchitecture levels. We currently only support `x86-64-v3`, `x86-64-v4`. You should only use `x86-64-v4` for those that supports AVX-512, such as all recent Intel Xeon CPUs or AMD Zen 4. While AMD's approach of double pumped AVX-512 implementation does not offers a factor of 2 speed up, but the availability of newer instructions from AVX-512 would still offers small amount of speed up. In short, `x86-64-v4` is recommended for AMD Zen 4.

MPI implementation
: MPICH:

> MPICH is one of the most popular implementations of MPI. It is used as the foundation for the vast majority of MPI implementations, including IBM MPI (for Blue Gene), Intel MPI, Cray MPI, Microsoft MPI, Myricom MPI, OSU MVAPICH/MVAPICH2, and many others. From MPICH - Wikipedia.

Open MPI is an alternative implementation of MPI. This is currently the only supported MPI implementation by us. See *CVMFS* and *OpenMPI with CVMFS* for examples.

Examples

To choose an environment for the tarball method where you want to develop the packages and update it, the simplest choice is https://github.com/ickc/so-software-environment/releases/download/20231214/tmp_so-conda-py310-20231214.tar.gz. Change the date `20231214` to whichever one you want (today's date for example).

To pick one from the CVMFS, you can see which ones are available first:

```
ls /cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/pmpm
```

The example given in *OpenMPI with CVMFS* uses `/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/pmpm/so-pmpm-py310-mkl-x86-64-v3-openmpi-latest`. Feel free to change the date here.

Automatic dispatch (Coming soon)

A wrapper will be created to auto-dispatch between `mkl` and `nomkl` versions, and between `x86-64-v3` and `x86-64-v4` versions based on the CPU of the worker nodes.

In this section we will focus on different methods to deploy software to the worker nodes.

Examples used in this section can also be found under `2-software-deployment/` in this repository relative to this file.

The *de facto* way to deploy software on CERN compatible grid system including Blackett where our data centre resides in is via CVMFS. Container technologies are also available, but HTCondor imposes a huge limitation as universes are mutually exclusive, and therefore you cannot put a job in docker/container universe while in parallel universe. As multi-nodes parallelism is important for CMB data analysis, we will not cover container here, although you are welcome to try as a developer. Note that CVMFS is mounted as read-only on the worker nodes, and can only deployed by people with elevated privilege. So CVMFS is for production only and is of limited use for development. Lastly, we will also mention a recommended way to deploy any softwares by packaging things in a tarball and transfer it to the worker nodes.

Deployment Method	Description	Suitable For	Limitations	Support level
Tarball	Packaging software and its dependencies in a compressed format, then transferring to worker nodes.	Custom software	Manual management; Transfer overhead.	Documentation provided to enable you as a developer to control your stack.
CVMFS	Standardized software distribution system used in CERN's grid.	Production software	Read-only; Needs elevated privilege for deployment.	Software deployment is centrally maintained, deployed periodically. You can requests softwares to be included, and approvals will be granted based on technical limitations.
Containers	Use of technologies like Docker to encapsulate software in isolated environments.	Development, Testing	Cannot be used with the `parallel` universe in HTCondor.	Not supported but you can feel free to try when multi-node parallelism is not needed.[1]

4.3 MPI Applications

4.3.1 MPI with a single node

Technically, HTCondor talks about machines rather than nodes, where a requested machine with a certain amount of CPU can be sharing the same physical node with other jobs.

In this example, we'd mention that you can run MPI applications using the vanilla universe in a single node. This is the simplest case as they do not need to communicate over multiple machines/nodes/HTCondor processes.

In `mpi.ini`,

```
universe                = vanilla
executable              = mpi.sh
should_transfer_files   = yes
when_to_transfer_output = ON_EXIT
```

(continues on next page)

[1] Blackett's support of container is copied from here as below:

- The docker universe won't work because there's no docker on the worker nodes.
- The worker nodes have Apptainer installed though. Jobs can use it to run their workloads inside a container.
- Running containers in a vanilla universe job works fine.
- No experience with doing the same in parallel universe jobs. There might be communication problems between the MPI processes if MPI is running inside a container.

Note that while Apptainer (formerly Singularity) is supported by Blackett to a certain capacity, this is not supported by us, the SO:UK Data Centre. Again, feel free to experiment with it, and even contribute to this documentation in the developer guide when you find it useful.

(continued from previous page)

```
transfer_input_files    = mpi.sh
request_cpus            = 16
request_memory          = 32999
request_disk            = 32G

log                     = mpi.log
output                  = mpi.out
error                   = mpi.err
stream_error            = True
stream_output           = True

queue
```

In `mpi.sh`,

```
#!/usr/bin/env bash

# helpers #####################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

###############################################################################

CONDA_PREFIX=/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/pmpm/so-pmpm-py310-mkl-
↪x86-64-v3-mpich-latest

# note that as you're not using the MPI wrapper script to launch in the parallel␣
↪universe,
# you need to set these environment variables for the hybrid MPI parallelization to␣
↪not be over-subscribed.
# rule of thumb is *_NUM_THREADS * no. of MPI processes should equals to the no. of␣
↪physical (not logical) cores
# i.e. 2 threads * 4 processes = 8 physical cores here
export OPENBLAS_NUM_THREADS=2
export JULIA_NUM_THREADS=2
export TF_NUM_THREADS=2
export MKL_NUM_THREADS=2
export NUMEXPR_NUM_THREADS=2
export OMP_NUM_THREADS=2

print_double_line
echo "$(date) activate environment..."
. "$CONDA_PREFIX/bin/activate"
print_line
echo "Python is available at:"
which python
echo "mpirun is available at:"
```

(continues on next page)

(continued from previous page)

```
which mpirun

print_double_line
echo 'Running TOAST tests in /tmp...'
cd /tmp
mpirun -n 4 python -c 'import toast.tests; toast.tests.run()'
```

It is this simple.

Note: We uses an environment from CVMFS here, where we will provide more details in *OpenMPI with CVMFS*.

We also uses MPICH in this case. Currently we only support Open MPI with the Parallel Universe. But in the Vanilla Universe, there's no such limitation as single node MPI is really that simple.

Lastly, submit the job as usual by

```
condor_submit mpi.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit mpi.ini; tail -F mpi.log mpi-0.out mpi-0.err mpi-1.out mpi-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

4.3.2 OpenMPI

Deploying MPI applications using the provided wrapper

Create a file `mpi.ini`:

```
universe                = parallel
executable              = /opt/simonsobservatory/cbatch_openmpi
arguments               = env.sh mpi.sh
machine_count           = 2
should_transfer_files   = yes
when_to_transfer_output = ON_EXIT
transfer_input_files    = env.sh,mpi.sh,/opt/simonsobservatory/pmpm-20230718-Linux-
↪x86_64-OpenMPI.tar.gz
request_cpus            = 16
request_memory          = 32999
request_disk            = 32G

log                     = mpi.log
output                  = mpi-$(Node).out
error                   = mpi-$(Node).err
stream_error            = True
stream_output           = True

queue
```

Note that it calls a wrapper script `cbatch_openmpi`. You can read the documentation of the script from the beginning of that script, copied below:

```
# this modifies from
# https://github.com/htcondor/htcondor/blob/main/src/condor_examples/openmpiscript

#* usage: cbatch_openmpi.sh <env.sh> <mpirun.sh>
#* this is refactor from openmpiscript to provide user more control over the␣
↪environment and the mpirun command to use.
#* the <env.sh> is a script that setup the environment, including OpenMPI. e.g.␣
↪contains `module load mpi/openmpi3-x86_64`
#* the <mpirun.sh> is a script that runs mpirun with the desired arguments,
#* this script can either setup the host on their own, or use the 2 convenience␣
↪functions provided below:
#* set_OMPI_HOST_one_slot_per_condor_proc setup one slot per condor process, useful␣
↪for hybrid-MPI
#* set_OMPI_HOST_one_slot_per_CPU setup one slot per CPU
#* then in the <mpirun.sh>, use `mpirun -host "$OMPI_HOST" ...`

#* other functionally different changes from openmpiscript:
#* remove redundant checks such as EXINT, _USE_SCRATCH
#* remove MPDIR and --prefix=... in mpirun: module load is sufficient
#* instead of generating a HOSTFILE and use mpirun --hostfile $HOSTFILE ..., use␣
↪mpirun --host $OMPI_HOST ... instead.
#* set OMPI_MCA_btl_base_warn_component_unused=0 to suppress warning about unused␣
↪network interfaces
#* remove chmod on executable, the user should have done this already
#* refactor the usage of the script, rather than openmpiscript MPI_EXECUTABLE ARGS ...
↪, use cbatch_openmpi.sh <env.sh> <mpirun.sh>. See above for documentation.
```

Note that this script takes 2 arguments, both are also scripts, where the 1st one setup the software environment and the second one runs the MPI application.

In the first file `env.sh`,

```
#!/bin/bash -l

# helpers #####################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

###############################################################################

print_double_line
echo "$(date) unarchive environment..."
tar -xzf pmpm-20230718-Linux-x86_64-OpenMPI.tar.gz -C /tmp

print_double_line
echo "$(date) activate environment..."
source /tmp/pmpm-20230718/bin/activate /tmp/pmpm-20230718
print_line
```

(continues on next page)

(continued from previous page)

```
echo "Python is available at:"
which python
echo "mpirun is available at:"
which mpirun
```

We see that it is basically preparing for the software environment following *this section*.

The reason this wrapper script has such an interface is because MPI is part of your software environment. Only after you loaded this environment (where you can change to any OpenMPI installation you want as long as it is OpenMPI), the wrapper script can continue to start the OpenMPI launcher to prepare for you to run `mpirun` later.

Then in `mpi.sh`,

```
#!/usr/bin/env bash

# helpers ############################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

######################################################################

print_double_line
set_OMPI_HOST_one_slot_per_condor_proc
echo "Running mpirun with host configuration: $OMPI_HOST" >&2

print_double_line
echo 'Running TOAST tests in /tmp...'
cd /tmp
mpirun -v -host "$OMPI_HOST" python -c 'import toast.tests; toast.tests.run()'
```

Here `set_OMPI_HOST_one_slot_per_condor_proc`, provided within the wrapper script, is called to set `OMPI_HOST`. There are 2 such bash functions provided. Be sure to read the `cbatch_openmpi` documentation to know which one to choose. The recommended setup for Hybrid MPI such as MPI+OpenMP is to use `set_OMPI_HOST_one_slot_per_condor_proc` such that each HTCondor process is one MPI process.

Note the use of `mpirun -host "$OMPI_HOST" ...`, which uses the prepared `OMPI_HOST` to launch the MPI processes.

> **Warning:** When writing these scripts such as `env.sh` and `mpi.sh`, note that in parallel universe in HTCondor, the `executable` is run in the single program, multiple data (SPMD) paradigm. This is very different from what you would do in SLURM's batch script for example.
>
> As a concrete example, if there's a line `echo hello world` in your scripts, in each HTCondor process, `echo hello world` will be run once, and there corresponding `mpi-?.out` files will each have a `hello world` there.
>
> So when `env.sh` is run, in each of the HTCondor process, the software environment is being preparing individually.

This is important, as all processes should shares exactly the same software environment to launch an MPI program.

Lastly, submit the job as usual by

```
condor_submit mpi.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit mpi.ini; tail -F mpi.log mpi-0.out mpi-0.err mpi-1.out mpi-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

Warning: It is known that when running the TOAST3 test suite with OpenMPI using our provided software environment has some failed unit tests. We are investigating and will be fixed in the future.

4.3.3 OpenMPI with CVMFS

This is similar to the previous example but with CVMFS instead.

Deploying MPI applications using the provided wrapper

Create a file `mpi.ini`:

```
universe                = parallel
executable              = /opt/simonsobservatory/cbatch_openmpi
arguments               = env.sh mpi.sh
machine_count           = 2
should_transfer_files   = yes
when_to_transfer_output = ON_EXIT
transfer_input_files    = env.sh,mpi.sh
request_cpus            = 16
request_memory          = 32999
request_disk            = 32G

# contraining CPU to match the environment using in env.sh
# Requirements          = (Arch == "INTEL") && (Microarch == "x86_64-v4")
# currently the only attributes that is exposed at Blackett is
Requirements            = Arch == "X86_64"

log                     = mpi.log
output                  = mpi-$(Node).out
error                   = mpi-$(Node).err
stream_error            = True
stream_output           = True

queue
```

Note that it calls a wrapper script `cbatch_openmpi`.

Note that this script takes 2 arguments, both are also scripts, where the 1st one setup the software environment and the second one runs the MPI application.

In the first file `env.sh`,

```
#!/bin/bash -l

# helpers #############################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

#######################################################################

CONDA_PREFIX=/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/pmpm/so-pmpm-py310-mkl-
↪x86-64-v3-openmpi-latest

print_double_line
echo "$(date) activate environment..."
source "$CONDA_PREFIX/bin/activate"
print_line
echo "Python is available at:"
which python
echo "mpirun is available at:"
which mpirun
```

We see that it is basically preparing for the software environment following *CVMFS*.

Note: See *Continuous Deployment (CD)* for tips on which CVMFS environment to choose.

The reason this wrapper script has such an interface is because MPI is part of your software environment. Only after you loaded this environment (where you can change to any OpenMPI installation you want as long as it is OpenMPI), the wrapper script can continue to start the OpenMPI launcher to prepare for you to run `mpirun` later.

Then in `mpi.sh`,

```
#!/usr/bin/env bash

# helpers #############################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
```

(continues on next page)

(continued from previous page)

```
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

################################################################################

print_double_line
set_OMPI_HOST_one_slot_per_condor_proc
echo "Running mpirun with host configuration: $OMPI_HOST" >&2

print_double_line
echo 'Running TOAST tests in /tmp...'
cd /tmp
mpirun -v -host "$OMPI_HOST" python -c 'import toast.tests; toast.tests.run()'
```

Here `set_OMPI_HOST_one_slot_per_condor_proc`, provided within the wrapper script, is called to set `OMPI_HOST`. There are 2 such bash functions provided. Be sure to read the `cbatch_openmpi` documentation to know which one to choose. The recommended setup for Hybrid MPI such as MPI+OpenMP is to use `set_OMPI_HOST_one_slot_per_condor_proc` such that each HTCondor process is one MPI process.

Note the use of `mpirun -host "$OMPI_HOST" ...`, which uses the prepared `OMPI_HOST` to launch the MPI processes.

Warning: When writing these scripts such as `env.sh` and `mpi.sh`, note that in parallel universe in HTCondor, the `executable` is run in the single program, multiple data (SPMD) paradigm. This is very different from what you would do in SLURM's batch script for example.

As a concrete example, if there's a line `echo hello world` in your scripts, in each HTCondor process, `echo hello world` will be run once, and there corresponding `mpi-?.out` files will each have a `hello world` there.

So when `env.sh` is run, in each of the HTCondor process, the software environment is being preparing individually. This is important, as all processes should shares exactly the same software environment to launch an MPI program.

Lastly, submit the job as usual by

```
condor_submit mpi.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit mpi.ini; tail -F mpi.log mpi-0.out mpi-0.err mpi-1.out mpi-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

Warning: It is known that when running the TOAST3 test suite with OpenMPI using our provided software environment has some failed unit tests. We are investigating and will be fixed in the future.

4.3.4 OpenMPI with CVMFS with AVX-512 CPU instructions

This is similar to the previous example but with AVX-512 instructions instead. Please refer to the last example for more explanations. Below we will only show the example.

Deploying MPI applications using the provided wrapper

Create a file `mpi.ini`:

```
universe                = parallel
executable              = /opt/simonsobservatory/cbatch_openmpi
arguments               = env.sh mpi.sh
machine_count           = 2
should_transfer_files   = yes
when_to_transfer_output = ON_EXIT
transfer_input_files    = env.sh,mpi.sh
request_cpus            = 16
request_memory          = 32999
request_disk            = 32G

# contraining CPU to match the environment using in env.sh
# Requirements          = (Arch == "INTEL") && (Microarch == "x86_64-v4")
# currently the only attributes that is exposed at Blackett is
Requirements            = has_avx512f && has_avx512dq

log                     = mpi.log
output                  = mpi-$(Node).out
error                   = mpi-$(Node).err
stream_error            = True
stream_output           = True

queue
```

In the first file `env.sh`,

```
#!/bin/bash -l

# helpers #################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

###########################################################################

CONDA_PREFIX=/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory/pmpm/so-pmpm-py310-mkl-
↪x86-64-v4-openmpi-latest
```

(continues on next page)

(continued from previous page)

```
print_double_line
echo "$(date) activate environment..."
source "$CONDA_PREFIX/bin/activate"
print_line
echo "Python is available at:"
which python
echo "mpirun is available at:"
which mpirun
```

Then in `mpi.sh`,

```
#!/usr/bin/env bash

# helpers ########################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}

print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

##################################################################

print_double_line
set_OMPI_HOST_one_slot_per_condor_proc
echo "Running mpirun with host configuration: $OMPI_HOST" >&2

print_double_line
echo 'Running TOAST tests in /tmp...'
cd /tmp
mpirun -v -host "$OMPI_HOST" python -c 'import toast.tests; toast.tests.run()'
```

Lastly, submit the job as usual by

```
condor_submit mpi.ini
```

After waiting for a while as the job finished, you can see what happened by reading the contents of `log`, `output`, and `error` as specified in the ClassAd.

See *Monitor your jobs* to see how to monitor the status of your job. For advance use, use this command instead,

```
condor_submit mpi.ini; tail -F mpi.log mpi-0.out mpi-0.err mpi-1.out mpi-1.err
```

and see *Streaming stdout & stderr with tail* for an explanation on what it does.

HTCondor's parallel universe is generic, in the sense that it is not designed around MPI specifically. Unlike other workload manager such as SLURM, the MPI launcher needs to be bootstrapped in the beginning of a parallel job. Therefore, a wrapper script is provided by us to encapsulate the MPI bootstrapping process. You are welcome to modify from the provided wrapper scripts to tailor for your specific workflows.

For now, OpenMPI is supported. We are investigating in making MPICH3+ works and eventually we would probably support MPICH3+ only.

4.4 Reading and writing data

4.4.1 Transferring files via ClassAd using HTCondor

We already saw how we can use ClassAd with HTCondor to transfer files from submit node to worker nodes.

For the other direction (from worker nodes back to submit node), you can specify this in the ClassAd to transfer output files, for example,

```
transfer_output_files = schedules,out_f090_i1_Jupiter
```

These comma-separated paths can be files or directories, and in case of directories, the entirety of the contents within will be transferred back to the submit node.

Note: This is not a recommended method to transfer large amount of data, as the submit node only has ~200GiB of local storage.

4.4.2 The grid storage system

User credentials

This part needed to be done once per machine.

You need to have your grid certificate `certBundle.p12` ready that you obtained from *this section*. Then run

```
mkdir -p "$HOME/.globus/"
mv certBundle.p12 "$HOME/.globus/usercred.p12"
chmod 600 "$HOME/.globus/usercred.p12"
```

Creating a proxy

Note: This part needed to be done periodically, it will expires in a week in the example given below.

```
voms-proxy-init --voms souk.ac.uk --valid 168:0
```

The command `voms-proxy-init` is used to contact the VOMS server and retrieve an Attribute Certificate (AC) containing user attributes that will be included in the proxy certificates.

The Attribute Certificate (AC) is configured with a maximum validity of 168 hours (7 days).

Example output after running this command will be:

```
$ voms-proxy-init --voms souk.ac.uk --valid 168:0
Enter GRID pass phrase for this identity:
Contacting voms03.gridpp.ac.uk:15519 [/C=UK/O=eScience/OU=Imperial/L=Physics/
↪CN=voms03.gridpp.ac.uk] "souk.ac.uk"...
Remote VOMS server contacted succesfully.


Created proxy in /tmp/x509up_u$UID.
```

(continues on next page)

(continued from previous page)

```
Your proxy is valid until Tue Nov 14 08:45:38 GMT 2023
```

The path `/tmp/x509up_u$UID` will be useful later. You can also run `voms-proxy-info --all` and see it again in the `path` attributes.

Accessing the grid storage system using GFAL

You can now access our grid storage system at

- davs://bohr3226.tier2.hep.manchester.ac.uk:443//dpm/tier2.hep.manchester.ac.uk/home/souk.ac.uk/, or
- root://bohr3226.tier2.hep.manchester.ac.uk:1094//dpm/tier2.hep.manchester.ac.uk/home/souk.ac.uk/.

Warning: Notice the double slash in `...ac.uk:...//dpm/...`. If a single slash is used, some tools might fail.

For example, to see what's inside,

```
gfal-ls -alH --full-time davs://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.
↪manchester.ac.uk/home/souk.ac.uk/
# or
gfal-ls -alH --full-time root://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.
↪manchester.ac.uk/home/souk.ac.uk/
```

And to make a directory there,

```
gfal-mkdir davs://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/
↪home/souk.ac.uk/testing
# or
gfal-mkdir root://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/
↪home/souk.ac.uk/testing
```

To delete it,

```
gfal-rm -r davs://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/
↪home/souk.ac.uk/testing
# or
gfal-rm -r root://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/
↪home/souk.ac.uk/testing
```

Note: We omitted the port when gfal is used here, as the default ports are used.

Warning: You can delete files created by others, vice versa. Thing twice before deleting or overwriting. To protect your files, you may use gfal-chmod below.

More info

As of writing, the versions of the softwares are

```
gfal-ls v1.7.1
GFAL-client-2.21.5
```

Available commands:

```
gfal2_version
gfal-evict
gfal-mkdir
gfal-sum
gfal-archivepoll
gfal-legacy-bringonline
gfal-rename
gfal-token
gfal-bringonline
gfal-legacy-register
gfal-rm
gfal-xattr
gfal-cat
gfal-legacy-replicas
gfal-save
gfal-chmod
gfal-legacy-unregister
gfal_srm_ifce_version
gfal-copy
gfal-ls
gfal-stat
```

Some of the commands mimics corresponding POSIX commands:

gfal-mkdir
: `mkdir`—Creates directories.

gfal-cat
: `cat`—Displays the content of a file.

gfal-chmod
: `chmod`—Changes file permissions and modes.

gfal-rm
: `rm`—Removes files or directories.

gfal-copy
: `cp`—Copies files and directories.

gfal-ls
: `ls`—Lists files and directories.

gfal-stat
: `stat`—Displays detailed information about files and directories.

gfal-rename
: `mv`—Renames or moves files and directories.

Check their respective man pages or help string for more information and see available options. For example, run

```
man gfal-ls
gfal-ls -h
```

Accessing the grid storage system from worker nodes

In the last section, we have seen how to connect to the grid storage system from a computer, including the submit nodes that we maintained.

We will now see how it can be accessed from within worker nodes.

If you haven't done already, you will need to setup the user-side access on our submit nodes by following the *User credentials* section. You will also need to run *Creating a proxy* periodically.

As usual, you can create a proxy using

```
voms-proxy-init --voms souk.ac.uk --valid 168:0
```

This will creates an Attribute Certificate (AC) to `/tmp/x509up_u$UID`.

Example job

From now on **we assumes you already created a proxy recently and it has not been expired**.

In `gfal.ini`, we set `use_x509userproxy`, and HTCondor will automatically copy from the standard location of the generated AC above and transfer it to the worker node for us.

```
executable              = gfal.sh

log                     = gfal.log
output                  = gfal.out
error                   = gfal.err

use_x509userproxy       = True

should_transfer_files   = Yes
when_to_transfer_output = ON_EXIT

request_cpus            = 1
request_memory          = 512M
request_disk            = 1G

queue
```

And in `gfal.sh`,

```
#!/bin/bash -l

# helpers #####################################################################

COLUMNS=72

print_double_line() {
        eval printf %.0s= '{1..'"${COLUMNS}"\}
        echo
}
```

(continues on next page)

(continued from previous page)

```
print_line() {
        eval printf %.0s- '{1..'"${COLUMNS}"\}
        echo
}

########################################################################

PROJ_DIR='bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/home/
↪souk.ac.uk'

for PROTOCOL in davs root; do
        print_double_line
        echo "Testing gfal-ls with $PROTOCOL"
        print_line
        gfal-ls -alH --full-time "$PROTOCOL://$PROJ_DIR"

        print_double_line
        echo "Testing gfal-mkdir with $PROTOCOL"
        gfal-mkdir -p "$PROTOCOL://$PROJ_DIR/$USER/testing"
        print_line
        gfal-ls -alH --full-time "$PROTOCOL://$PROJ_DIR/$USER"

        print_double_line
        echo "Testing gfal-rm with $PROTOCOL"
        print_line
        gfal-rm -r "$PROTOCOL://$PROJ_DIR/$USER/testing"

        print_double_line
        echo "Testing gfal-copy with $PROTOCOL"
        echo "hello $PROTOCOL" > "hello-$PROTOCOL.txt"
        gfal-copy -f "hello-$PROTOCOL.txt" "$PROTOCOL://$PROJ_DIR/$USER"
done
```

Note that any gfal commands from *this section* can be used so that you can either copy files from the grid storage system to the worker nodes in the beginning of your script, or copy files from the current worker node to the grid storage system by the end of your script.

Lastly, submit and see what happens[1]

```
condor_submit gfal.ini; tail -F gfal.log gfal.out gfal.err
```

After the job finished, you can check your output files copied to the grid storage system, like so

```
gfal-ls davs://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/
↪home/souk.ac.uk/$USER/
gfal-cat davs://bohr3226.tier2.hep.manchester.ac.uk//dpm/tier2.hep.manchester.ac.uk/
↪home/souk.ac.uk/$USER/hello-davs.txt
...
```

[1] See *Streaming stdout & stderr with tail* for an explanation on what the `tail` command does.

Mounting the grid storage system as a POSIX filesystem

You can mount the grid storage system as a POSIX filesystem on our login (submit) node, or any other computer if you have setup the VOMS client correctly following *this section*.

This is for ease of interactive use, for example if you want to quickly see what files are there, or use command line programs such as `ranger`, `tree`, etc. that expects a POSIX filesystem. You could also run some lightweight scripts over the filesystem, but note that this usage is not performant and discouraged.

Mounting via `xrootdfs`

A wrapper script is provided at `/opt/simonsobservatory/xrootdfs.sh` on `vm77`, and also in this repository.

If you haven't done already, you will need to setup the user-side access on our submit nodes by following the *User credentials* section. You will also need to run *Creating a proxy* periodically.

As usual, you can create a proxy using

```
voms-proxy-init --voms souk.ac.uk --valid 168:0
```

This will creates an Attribute Certificate (AC) to `/tmp/x509up_u$UID`.

Once your have this AC set up, you can run `/opt/simonsobservatory/xrootdfs.sh start` to mount it to `~/souk.ac.uk`, and `/opt/simonsobservatory/xrootdfs.sh stop` to unmount it.

> **Warning:** Once your AC expired, you need to go through this section again to re-generate the AC, and run `/opt/simonsobservatory/xrootdfs.sh restart`.

Feel free to modify the wrapper script, as copied below:

```
#!/usr/bin/env bash

# modified from https://github.com/xrootd/xrootd/blob/master/src/XrdFfs/xrootdfs.
↪template

# chkconfig: 345 99 10
# chkconfig(sun): S3 99 K0 10
# description: start and stop XrootdFS

MOUNT_POINT1="$HOME/souk.ac.uk"

start() {
    mkdir -p "$MOUNT_POINT1"
    # export XrdSecPROTOCOL=gsi
    # export X509_USER_PROXY="/tmp/x509up_u$$UID"
    # export XrdSecGSICREATEPROXY=0

    # we need to load the fuse kernel module
    /sbin/modprobe fuse
    ulimit -c unlimited
    cd /tmp

    # Please repeat the following lines for each additional mount point.

    # XROOTDFS_RDRURL is a ROOT URL to tell XrootdFS which base path should be␣
```

(continues on next page)

(continued from previous page)

```
↪mounted.
    XROOTDFS_RDRURL='root://bohr3226.tier2.hep.manchester.ac.uk:1094//dpm/tier2.hep.
↪manchester.ac.uk/home/souk.ac.uk/'

    # After XrootdFS starts but before it takes any user request, XrootdFS will try␣
↪to switch its effective
    # user ID to XROOTDFS_USER if it is defined.
    # export XROOTDFS_USER='daemon'
    # export XROOTDFS_USER="$USER"

    # if you are ready to use 'sss' security module. See README for detail
    # export XROOTDFS_SECMOD='sss'

    # XROOTDFS_CNSURL
    # XROOTDFS_FASTLS
    # XROOTDFS_OFSFWD
    # XROOTDFS_NO_ALLOW_OTHER

    # xrootdfs "$MOUNT_POINT1" -o allow_other,fsname=xrootdfs,max_write=131072,attr_
↪timeout=10,entry_timeout=10
    xrootdfs "$MOUNT_POINT1" -o rdr="$XROOTDFS_RDRURL"
    # undefine them so that the next mount point won't be affected by the setting for␣
↪the previous mount point.
    # unset XROOTDFS_RDRURL
    # unset XROOTDFS_USER
    # unset XROOTDFS_SECMOD
}
stop() {
    # repeat the following lines for each additional mount point
    # umount $MOUNT_POINT1
    fusermount -u "$MOUNT_POINT1"
}

case "$1" in
start)
    start
    ;;

stop)
    stop
    ;;

restart)
    stop
    sleep 5
    start
    ;;

*)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
esac
```

We will now look at the grid storage system, where the bulk of our data will resides in. We will look at

1. Initial setup,
2. accessing the grid storage system in HTCondor jobs on worker nodes, and

3. mount it as a POSIX filesystem in our login (submit) node for interactive use.

4.4.3 Librarian Overview

Librarian keeps track of all of the primary data products stored at a given site. There is a Flask-based server that keeps track of everything using a database and presents a nice frontend, and Python client code that can make various requests of one or more servers.

Reading data from Librarian (Coming soon)

For reading data, we already see how you can transfer your job scripts and any files from ClassAd. Here we will provide one more option for you to load SO specific Librarian Books.

For writing data, we will see how you can transfer any output files from worker nodes in configuring your ClassAd. But this has a limitation that our submit node currently only has ~200GiB and is not suitable to write large amount of files there.

Then we will talk about the *de facto* choice to write large amounts of output files on such grid system—the grid storage system.

4.5 Interactive sessions

4.5.1 Interactive job

To request an interactive job in the vanilla universe, create a file `example.ini`,

```
RequestMemory     = 32999
RequestCpus       = 16
use_x509userproxy = True
queue
```

Here, we assume the use of the grid storage system. Following *User credentials*, you should run

```
voms-proxy-init --voms souk.ac.uk --valid 168:0
```

to ensure your temporary AC is valid.

And then submit your job using

```
condor_submit -i example.ini
```

After submitting and waiting for a while, you will be dropped into an interactive bash shell on an interactive worker node.

Note: Interactive node is in the vanilla universe in HTCondor, meaning that you cannot requests multiple nodes here.

Note: The interactive job started in a worker node is in a blank state. I.e. it does not see the same `HOME` as in the submit node `vm77`. Any of the software deployment methods or data I/O mentioned earlier can be applied here, such as the grid storage system and CVMFS.

Explanation

`RequestMemory=32999`
: This line specifies that the job requires 32,999 megabytes (or roughly 32.999 gigabytes) of memory. The HTCondor system will attempt to match this job with a machine that has at least this much available memory.

`RequestCpus=16`
: This line indicates that the job needs 16 CPUs (or cores). Again, HTCondor will try to find a machine that can provide this number of CPUs for the job.

> **Warning:** By default, these are number of logical cores. Because of Simultaneous multithreading (SMT), this usually means the no. of physical cores is half of this number. This will have important consequence on over-subscription that we will mention later.

`use_x509userproxy = True`
: HTCondor will automatically transfer your AC in submit node to the worker node and set it up correctly such that you can access the grid storage system on this interactive worker node as well.

`queue`
: This line is a command that tells HTCondor to queue the job. When you submit this ClassAd, the job will be added to the queue and HTCondor will try to find a suitable machine that meets the specified requirements.

In this section, we will see how we can request worker node for interactive use.

The typical life-cycle of a workflow / pipeline are

1. Write a job configuration file and submit it to the workload manager,
    - which typically include a job script, say written in bash, as well.
2. Bootstrap a software environment on the worker node.
3. (Optionally) launch MPI applications using the provided wrapper in parallel universe.
4. I/O:
    - Reading data, as the input of your application,
    - Writing and storing data, as the product of your application.

In the following sections, we will go through these one by one. Per stage, there exists multiple solutions or situations that we will go through. In the end, we will provide example workflows where each touches all these aspects.

CHAPTER

FIVE

SO:UK DATA CENTRE SYSTEMS

5.1 System specifications

Currently, our testbed includes the following nodes. Note that some nodes can be submitted via vanilla universe only, parallel universe only, or both.

	CPU model	CPU vendor	CPU generation	CPU microarchitecture	No. of sockets	Total no. of logical cores	Total no. of physical cores	Total memory (GiB)	Total swap (GiB)	vanilla universe	parallel universe
hostname											
wn3805340.tier2.hep.manchester.ac.uk	Intel® Xeon® CPU E5-2620 v4 @ 2.10GHz	GenuineIntel	broadwell	x86_64_	2	16	16	63	64	True	False
wn3805341.tier2.hep.manchester.ac.uk	Intel® Xeon® CPU E5-2620 v4 @ 2.10GHz	GenuineIntel	broadwell	x86_64_	2	16	16	63	64	True	False
wn3806200.tier2.hep.manchester.ac.uk	Intel® Xeon® CPU E5-2620 v4 @ 2.10GHz	GenuineIntel	broadwell	x86_64_	2	32	16	63	64	False	True
wn3806201.tier2.hep.manchester.ac.uk	Intel® Xeon® CPU E5-2620 v4 @ 2.10GHz	GenuineIntel	broadwell	x86_64_	2	32	16	63	64	False	True
wn3806240.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	True	True
wn3806241.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	True	True
wn3806250.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	True	True
wn3806251.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	True	True
wn3806290.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	False	True
wn3806291.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	False	True
wn3806300.tier2.hep.manchester.ac.uk	Intel® Xeon® Gold 6130 CPU @ 2.10GHz	GenuineIntel	skylake_a	x86_64_	2	64	32	187	64	False	True

5.1.1 Constraining jobs to run on a subset of available nodes

This information can be used to write ClassAd constraints. For example,

```
requirements = (machine == "wn1905340.in.tier2.hep.manchester.ac.uk`") || (machine ==
↪"wn5914090.in.tier2.hep.manchester.ac.uk`") || (machine == "wn5916090.in.tier2.hep.
↪manchester.ac.uk`") || (machine == "wn5916340.in.tier2.hep.manchester.ac.uk`") ||␣
↪(machine == "wn5914340.in.tier2.hep.manchester.ac.uk`") || (machine == "wn5917090.
↪in.tier2.hep.manchester.ac.uk`")
```

5.2 Introduction to CVMFS

C.f. *CVMFS*.

Properties of CVMFS:

- POSIX read-only file system
- user space (FUSE module)
- shares a universal namespace under `/cvmfs/...`
- world readable
- content managers: only special users who registered as content managers can write to this space (See *Register as a new content manager (CVMFS)* and *Publishing to CVMFS*.)
- synchronization lag: it takes some time for it to be synchronized from the publishing node. Empirically, it takes around 2.5-3 hours for contents to be sync'd on Blackett.
- world-readable: Anyone can see the contents so no secrets should be placed here.

5.3 The upcoming upgrade to Ceph & CephFS

Transition to CephFS is scheduled in early 2024.

CephFS will provide a similar experience to NERSC's SCRATCH space powered by LUSTRE. This will replace the clunky grid storage system for data I/O currently powered by DPM with XRootD and xrootdfs. This will also replace CVMFS for software deployment.

See transition between pre-production CephFS to production CephFS · Issue #24 · simonsobs-uk/data-centre.

- Blackett is currently using DPM and will transition into Ceph in 2024.
- Currently, XRootD is serving from DPM, which will transition into serving from Ceph instead.
- Ceph exposes the filesystem via CephFS (currently xrootdfs can be used to mount XRootD as FS via FUSE, but not on worker nodes), available on all nodes including worker nodes.
- pre-production CephFS will be put on our machines (the SO:UK DC purchased machines), but may be subject to purging when transitioning into production
  - This will not be a problem for us (stopping us using pre-production CephFS ASAP), and we plan to copy data for our users. Both can be mounted, and then say we make the pre-production CephFS read-only, and replicates it to the production CephFS. We should provide env. var. to hide the differences in absolute path (see #23)

Link:

- Ceph Deployment and Monitoring at Lancaster, GridP47, 23 March 2022, Gerard Hand, Steven Simpson, Matt Doidge

    There's only so many times you can declare a million files lost and not rethink all your life choices.

5.4 Worker nodes at Blackett

As the SO:UK Data Centre is part of Blackett, we have access to its Tier-2 services. This means by default when we submit jobs from HTCondor, we can have access to the facility as well, where its main users are from ATLAS and LHCb.

For more details of our current testbed, see *System specifications*.

5.4.1 Wall-clock time

Copied from Feedback on user documentation · Issue #14 · simonsobs-uk/data-centre.

- There's a maximum wall-clock time configured. It's currently **72 hours**, jobs get killed if they exceed this.
- The same is true for CPU time for each requested CPU (ie job gets killed if `total used CPU hours > requested CPUs * 72 hours`)
- Need to check how the machine count fits into this, we'll most likely have to update the configuration to take this into account (ie `machine count * requested CPUs * 72 hours`).

5.4.2 `/tmp` directory

Different HTCondor processes sees a different `/tmp` directory, even when they lands on the same physical nodes. `/tmp` is actually a symlink to somewhere within scratch there, and scratch is unique to each HTCondor process.

I.e. you do not need to worry our tarball method which rely on `/tmp` will clashes with other HTCondor process.

5.5 SO:UK Data Centre worker nodes (Coming Soon)

Note: Unless stated otherwise, as SO:UK Data Centre is part of Blackett, everything from the last section should applies here. SO:UK-Data-Centre-worker-nodes specific details and differences will be put here.

5.6 Submit node

Our submit node is `vm77`, a single, small VM instance for submitting jobs. This is where you ssh into.

5.7 JupyterHub (Coming Soon)

See Establishing a Dedicated Login Node for Interactive Computing and JupyterHub Integration · Issue #31 · simonsobs-uk/data-centre.

This is intended to be read by typical users following our recommended usage. While it is written for general audiences, some pointers specific to NERSC users will be given to adapt their pipelines at NERSC to the SO:UK Data Centre.

We will start by pointing out main differences between NERSC and SO:UK Data Centre that will have important implications to how to deploy workflows here.

Facility	NERSC	SO:UK Data Centre
Nature	HPC	HTC
Configuration	Homogeneous within a pool	Heterogeneous by default
Workload Manager	SLURM	HTCondor
Job Classification Model	Different QoS can be selected, such as debug, interactive, regular, premium, etc., categorized by priority and charge factors. They shares exactly the same software environments.	Different "universe" can be selected, like vanilla, parallel, docker, container, etc., based on software environments and job launch methods. Universes are mutually exclusive and hence a job cannot be configured to multiple universes simultaneously. Interactive job is only available in vanilla universe.
Login Node Designation	Login nodes reachable via ssh with 2-factor authentication. Passwordless login can be achieved by using `sshproxy` service to create temporary ssh keys.	Called Submit Node in HTCondor. Tentatively, a special login node named `vm77` is reachable via ssh. Users are required to submit ssh keys to maintainer, and is passwordless by default.
Compute Node Designation	Compute nodes	Worker nodes
Home Directory	Globally mounted home directory, backed up periodically	Not available on worker nodes
Archive Filesystem	HPSS	Not available
Scratch Filesystem	Parallel distributed file system (LUSTRE) with all SSD. Purged once every few months.	Local to each worker node. Data doesn't persist post job completion.
Software Distribution Filesystem	Read-only global common	Read-only CVMFS
Large Storage Pool	CFS with a filesystem interface	Grid storage system without a filesystem interface
Job Configuration	SLURM directives within the batch script	ClassAd in a separate, ini-like format
Wallclock Time	Must be specified in job configuration	Not applicable
Sharing Physical Nodes	Requested via interactive QoS	Always shared by default
Exclusive Physical Node Alloca-	Requested via regular QoS	Not applicable

CHAPTER

SIX

DEVELOPER GUIDE

6.1 HTCondor Glossary

ClassAd
ClassAd stands for "Classified Advertisement." It's a flexible and expressive language used by HTCondor for representing jobs, machines, and other resources. ClassAds are similar to attribute-value pairs, and they're used to match jobs with appropriate resources.

Scheduler
The component in HTCondor that queues and manages users' job submissions. It finds appropriate matches for the jobs in the queue using the ClassAd system.

Startd (Start Daemon)
This is the daemon running on the worker node that advertises the node's resources and capabilities. It's responsible for executing and managing jobs on the node.

Negotiator
Part of the central manager services, it is responsible for making match decisions, pairing submitted jobs with suitable execution resources.

Worker Nodes (or Execute Nodes)
The computational resources or machines where the jobs are executed. Each worker node runs a startd process.

Submit Node
The machine or location from which jobs are submitted into the HTCondor system. The scheduler runs on this node.

Central Manager
The central authority in an HTCondor pool that hosts the Negotiator and the Collector services. It's essential for resource matchmaking and information gathering.

Collector
A service running on the Central Manager that gathers ClassAd information from other daemons (like startd and schedd) across the pool.

Condor Pool
A collection of machines working under a single HTCondor system. This includes the Central Manager, Worker Nodes, and potentially multiple submit nodes.

Universe
In HTCondor, a Universe is a specific execution environment for a job. Examples include the Vanilla Universe, Parallel Universe, and Docker Universe. The chosen Universe determines how a job is executed and what features are available to it.

Checkpointing
A feature that allows jobs to be paused and resumed. This is especially useful if a job gets preempted or if the machine it's running on goes down.

Preemption
The act of suspending or stopping a currently running job to free up resources for another job that has higher priority or better matches the resources.

Rank
An expression in the ClassAd system that indicates a preference for a match. For example, a job might rank execution machines by available memory, favoring matches with more memory.

Requirements
Expressions in the ClassAd system that must be satisfied for a match to occur. If a job's requirements do not match the attributes of a machine, then the job will not be sent to that machine.

Dedicated Scheduling
In the HTCondor Parallel Universe, "dedicated" scheduling refers to the process by which certain compute nodes (machines) are reserved exclusively for running parallel jobs. Such a setup ensures that parallel jobs, like MPI jobs, have consistent and predictable communication between the nodes without interference from other non-parallel jobs. Dedicated scheduling is advantageous for jobs that require tight inter-process communication or a specific arrangement of nodes. When machines are part of the dedicated scheduler, they won't execute other tasks outside of the designated parallel jobs.

6.2 The grid storage system—revisited

6.2.1 Setting up Grid Community Toolkit (GCT) (Under construction)

> **Warning:** Work in-progress. For now, instruction to access the grid storage system on unsupported machines are not complete.

To authenticate your machine to the grid, you need to

```
# Arch Linux from AUR using yay
yay -S gct
```

Requesting a host certificate for your machine, go to https://portal.ca.grid-support.ac.uk/cert_owner/requestHostCert.

6.2.2 Setting up VOMS Clients

If you work on machines supported by us, you can skip this part. This part needed to be done once per machine.

> **Warning:** Work in-progress. For now, instruction to access the grid storage system on unsupported machines are not complete.

Installing the clients

You would need to install VOMS Clients by following the instruction in the VOMS Clients guide. The currently supported OS is either RHEL 6/7 or Debian 6 as of writing:

```
# RHEL
sudo yum install voms-clients-java
# Debian
sudo apt-get install voms-clients3
# Arch Linux from AUR using yay
yay -S voms-clients
```

Configuring VOMS trust anchors

Create these paths with the corresponding contents below:

```
# generated by running
$ head /etc/grid-security/vomsdir/souk.ac.uk/*
==> /etc/grid-security/vomsdir/souk.ac.uk/voms02.gridpp.ac.uk.lsc <==
/C=UK/O=eScience/OU=Oxford/L=OeSC/CN=voms02.gridpp.ac.uk
/C=UK/O=eScienceCA/OU=Authority/CN=UK e-Science CA 2B

==> /etc/grid-security/vomsdir/souk.ac.uk/voms03.gridpp.ac.uk.lsc <==
/C=UK/O=eScience/OU=Imperial/L=Physics/CN=voms03.gridpp.ac.uk
/C=UK/O=eScienceCA/OU=Authority/CN=UK e-Science CA 2B

==> /etc/grid-security/vomsdir/souk.ac.uk/voms.gridpp.ac.uk.lsc <==
/C=UK/O=eScience/OU=Manchester/L=HEP/CN=voms.gridpp.ac.uk
/C=UK/O=eScienceCA/OU=Authority/CN=UK e-Science CA 2B
```

Configuring VOMS server endpoints

Create these paths with the corresponding contents below:

```
# generated by running
$ head /etc/vomses/*
==> /etc/vomses/souk.ac.uk-voms02.gridpp.ac.uk <==
"souk.ac.uk" "voms02.gridpp.ac.uk" "15519" "/C=UK/O=eScience/OU=Oxford/L=OeSC/
↪CN=voms02.gridpp.ac.uk" "souk.ac.uk" "24"

==> /etc/vomses/souk.ac.uk-voms03.gridpp.ac.uk <==
"souk.ac.uk" "voms03.gridpp.ac.uk" "15519" "/C=UK/O=eScience/OU=Imperial/L=Physics/
↪CN=voms03.gridpp.ac.uk" "souk.ac.uk" "24"

==> /etc/vomses/souk.ac.uk-voms.gridpp.ac.uk <==
"souk.ac.uk" "voms.gridpp.ac.uk" "15519" "/C=UK/O=eScience/OU=Manchester/L=HEP/
↪CN=voms.gridpp.ac.uk" "souk.ac.uk" "24"
```

Jump to *User credentials* to continue your setup.

6.2.3 Installing Grid File Access Library (GFAL)

Unfortunately, the documentation at GFAL2 · Data Management Clients Documentation does not indicate how to install it. You could compile it yourself following Data Management Clients / gfal2 · GitLab, or on RHEL:

```
sudo yum install 'gfal2*' 'python3-gfal2*'
```

Other package managers might support it. Please provide a pull request / issue helping us to document this.

Move on to *Accessing the grid storage system using GFAL* to see how to use it.

6.2.4 Installing XRootD

Follow their documentation at xrootd/xrootd: The XRootD central repository.

Now follow *Mounting the grid storage system as a POSIX filesystem* or *Advanced usage: use XRootD to interact with the grid storage system directly* on how to use it.

6.2.5 Advanced usage: use XRootD to interact with the grid storage system directly

Rather than using GFAL, you could access via the XRootD protocol directly. For example,

```
XROOTDFS_RDRURL='root://bohr3226.tier2.hep.manchester.ac.uk:1094//dpm/tier2.hep.
↪manchester.ac.uk/home/souk.ac.uk/'
echo 'Hello world' > /tmp/test
xrdcp /tmp/test "$XROOTDFS_RDRURL/$USER-test"
rm -f /tmp/test
```

And it can also be used interactively, which provides a POSIX filesystem-like experience:

```
# this command enters an interactive mode
$ xrdfs bohr3226.tier2.hep.manchester.ac.uk
# how you can ls
[bohr3226.tier2.hep.manchester.ac.uk:1094] / > ls
/SRR
/atlas
/bes
/biomed
/cms
/dteam
/dune
/euclid.net
/eucliduk.net
/fermilab
/gridpp
/hone
/icecube
/ilc
/lhcb
/lsst
/lz
/ops
/pheno
/skatelescope.eu
/souk.ac.uk
```

(continues on next page)

(continued from previous page)

```
/t2k.org
/vo.northgrid.ac.uk
# or cd
[bohr3226.tier2.hep.manchester.ac.uk:1094] / > cd souk.ac.uk
[bohr3226.tier2.hep.manchester.ac.uk:1094] /souk.ac.uk > ls
/souk.ac.uk/erosenberg
```

Definitions

xrdcp
: This command is akin to the POSIX `cp` command. It's used for copying files and directories within XRootD or between XRootD and local file systems.

xrdfs
: This command can be compared to various POSIX file system commands. It allows users to interact with a remote file system using operations similar to those found in POSIX, like listing directories, creating/removing files, etc.

We will revisit the grid storage system again for advanced use. See *The grid storage system* from the User guide for simple uses. We will look at

1. How to setup the access to the grid storage system from other computers,
2. and other advanced usages.

6.3 Tips & tricks

6.3.1 Tips and gotchas when writing HTCondor ClassAds

Automatic detection of output files to transfer back to submit notes

If `should_transfer_files = YES` is specified, HTCondor has heuristics to automatically transfer **some** files created on the compute node back to your submit node, which can be a surprise to the end users.

`should_transfer_files = NO` is not the best choice however, for *this other reason*.

The recommended setup to suppress this behavior is:

```
should_transfer_files = YES
transfer_output_files = ""
# optionally, also
when_to_transfer_output = ON_SUCCESS
```

I.e. either specifically set `transfer_output_files` to the list of files and/or directories you need to transfer, or set it to empty string explicitly.

For more details, see Specifying What Files to Transfer — HTCondor Manual.

Setting `should_transfer_files = No` would prevent jobs from running on some nodes under certain circumstances

From issue #45:

It looks like HTCondor adds some restrictions to the requirements expression depending on the value of `should_transfer_files`:

- `IF_NEEDED` (that's also the default):

```
((TARGET.FileSystemDomain == MY.FileSystemDomain) || (TARGET.HasFileTransfer))
```a
```

- `YES`:

```
(TARGET.HasFileTransfer)
```

- `NO`:

```
(TARGET.FileSystemDomain == MY.FileSystemDomain)
```

As we don't have a shared filesystem, all nodes in the cluster have a different value for `MyFileSystemDomain`, it's set to the FQDN of each node. This will change once we have a shared filesystem in place.

6.3.2 Job managements

Monitor your jobs

In HTCondor, you can use

```
condor_q
```

to see the status of your job.

You can also watch them, like so

```
watch -n 15 condor_q
```

where 15 is the number of seconds you want the command to be repeated, i.e. you are monitoring the status once every 15s.

There is also a command from HTCondor which do the watching automatically:

```
condor_watch_q
```

Both approach has their pros and cons and you are welcome to try which one suits your purpose better.
```

Killing your jobs

After you submitted a job, you obtained a job id. You can also retrieve this job id from the `condor_q` command above. You can kill this job by using,

```
condor_rm $ID
# such as...
condor_rm 1983
```

Or you can also kill all jobs that is still in the queue created by you:

```
condor_rm $USER
```

Warning: This is destructive! All your submitted jobs will disappear and there's no way to bring them back except by resubmitting all of them again.

6.3.3 Streaming stdout & stderr with tail

When submitting a job in HTCondor (and any other computing facilities), often your job will be run on another node at a later time. If you eager to look at the output (stdout & stderr) as soon as it is running, HTCondor provided a facility to do that together with some UNIX utilities.

Firstly, HTCondor has a facility to stream the stdout & stderr from the worker nodes back to the submit node you are working on. To use a specific example from *Vanilla universe*,

```
...
output = hello_world.out
error = hello_world.err
log = hello_world.log
stream_output = True
stream_error = True
...
```

The `stream_output` & `stream_error` instructs the job to stream the stdout & stderr back to your submit node in real time (which would normally be transferred back only after the job terminates).

If we submit the job and run the `tail` command at once, like this

```
condor_submit example.ini; tail -F hello_world.log hello_world.out hello_world.err
```

Then the UNIX command `tail` would *follow* the files listed (which are the `output`, `error` and `log` specified in your ClassAd) as soon as new contents are available.

Detailed explanations

As an example, the output would looks something like

```
$ condor_submit example.ini; tail -F hello_world.log hello_world.out hello_world.err
Submitting job(s).
1 job(s) submitted to cluster 511.
```

which is the stdout from `condor_submit example.ini`. Then

```
==> hello_world.log <==
```

is the tail command working immediately to follow contents of `hello_world.log`, with the following contents:

```
000 (511.000.000) 2023-08-29 23:39:35 Job submitted from host: <195.194.109.199:9618?
↪addrs=195.194.109.199-9618+[2001-630-22-d0ff-5054-ff-fe9a-b662]-9618&alias=vm77.
↪tier2.hep.manchester.ac.uk&noUDP&sock=schedd_2377818_f2b3>
...
```

Then

```
tail: cannot open 'hello_world.out' for reading: No such file or directory
tail: cannot open 'hello_world.err' for reading: No such file or directory
```

is `tail` telling us that `hello_world.out` & `hello_world.err` does not exist yet, as the job hasn't started. `tail` will follow them as soon as they are available. Then `hello_world.log` continues to have more content, indicating its progress:

```
==> hello_world.log <==
040 (511.000.000) 2023-08-29 23:39:35 Started transferring input files
        Transferring to host: <195.194.109.209:9618?addrs=195.194.109.209-9618+[2001-
↪630-22-d0ff-5054-ff-fee9-c3d]-9618&alias=vm75.in.tier2.hep.manchester.ac.uk&noUDP&
↪sock=slot1_4_1883_7e66_41406>
...
040 (511.000.000) 2023-08-29 23:39:35 Finished transferring input files
...
```

Then

```
tail: 'hello_world.out' has appeared;  following end of new file
tail: 'hello_world.err' has appeared;  following end of new file
```

tells us that these files finally appeared (as the job has started). Then

```
001 (511.000.000) 2023-08-29 23:39:36 Job executing on host: <195.194.109.209:9618?
↪addrs=195.194.109.209-9618+[2001-630-22-d0ff-5054-ff-fee9-c3d]-9618&alias=vm75.in.
↪tier2.hep.manchester.ac.uk&noUDP&sock=startd_1389_5123>
...
```

continues to show more log from `hello_world.log`. This part

```
==> hello_world.out <==
hello world
```

Is the content of `hello_world.out` as soon as it appears, where in the end it has the following log:

```
==> hello_world.log <==
006 (511.000.000) 2023-08-29 23:39:36 Image size of job updated: 35
        0  -  MemoryUsage of job (MB)
        0  -  ResidentSetSize of job (KB)
...
005 (511.000.000) 2023-08-29 23:39:36 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
```

(continues on next page)

(continued from previous page)

```
        0  -  Run Bytes Sent By Job
        33088  -  Run Bytes Received By Job
        0  -  Total Bytes Sent By Job
        33088  -  Total Bytes Received By Job
        Partitionable Resources :    Usage  Request Allocated
           Cpus                 :                1         1
           Disk (KB)            :       44       35    832179
           Memory (MB)          :        0        1       100

        Job terminated of its own accord at 2023-08-29T22:39:36Z.
...
```

You will notice that the `tail` process has never ended, as if it is halting. The reason is that you are not looking at the output of the job itself, but monitoring the streaming output from the job via `tail`. As far as `tail` is concerned, it will continue to monitor (follow) any new contents from these 3 files and print it on your screen.

From the content itself, you see `Job terminated of its own accord...` meaning that your job has ended, and you should now press `Ctrl + c` to terminate the `tail` command.

You can also checkout *Monitor your jobs* to see how to monitor the status of your job, and from it you can tell this job has indeed ended.

In this section, we would go through some useful tips and tricks that eases the user experience of SO:UK Data Centre including HTCondor specific ones and some specific to our deployment at Blackett.

This section is sorted alphabetically and can mostly be read on its own without earlier sections.

This is intended to be read by advanced users who have the technical know-how to generalize the tips and pointers given here to optimize their applications.

CHAPTER

SEVEN

MAINTAINER GUIDE

7.1 Managing and Maintaining Computing Resources

7.1.1 New accounts

New SO:UK VO membership

Go to VOMS Admin > souk.ac.uk and manage there.

Adding new users on the submit node `vm77`

Point the user to *this section* and ask them to send those info to you.

0. If the user ssh key comment contains their email address, replace it with something else.
1. Edit `/usr/local/etc/staged/common.yaml` on `vm77`, replace the following `ALL_CAP` variables accordingly,

```
common::users:
  USER_NAME:
    uid: UID
    comment: FIRST_NAME LAST_NAME
    home: /home/USER_NAME
    keys:
      - "ssh-ed25519 KEY_VALUE COMMENT"
    password: "$6$...$..."
    groups:
      - sudo
      - simonsobservatory
```

> **Note:** For `UID`, increments according to the list. Just make sure it has not been used.
>
> For `sudo` groups, obviously only grant those you want them to have sudo privillege.

2. Tell Robert to update.
3. To check if the config is populated, check the file `/etc/passwd` and see if the new users is there. If it does, the user should be ready to go.

Warning: The content of this file contains sensitive information, such as salted, hashed passwords, which is configured to be readable only to `root` at `/etc/shadow`. Hence, the config file `/usr/local/etc/staged/common.yaml` should be treated with the same level of permission.

Register as a new content manager (CVMFS)

1. Run `ssh northgridsgm@cvmfs-upload01.gridpp.rl.ac.uk` from any computer. Then you'd see something like

   ```
   (northgridsgm@cvmfs-upload01.gridpp.rl.ac.uk) Authenticate at
   -----------------
   https://aai.egi.eu/device?user_code=...
   -----------------
   Hit enter when you have finished authenticating
   ```

   Follow the link and register there.

2. On https://aai.egi.eu/auth/realms/egi/account/#/personal-info, copy the content of "Username" field.
3. Follow CVMFS - GridPP Wiki to send an email including the username above as your `voperson_id`:

- Name of the VO or CVMFS repository: `northgrid.gridpp.ac.uk`
- The "voperson_id" from your account in EGI CheckIn: …@egi.eu

4. Wait for email from lcg-support@gridpp.rl.ac.uk when the admin added you to the service.

7.1.2 Software deployments

Publishing to CVMFS

See *above* if you haven't applied for the role of content manager yet.

Then

```
ssh northgridsgm@cvmfs-upload01.gridpp.rl.ac.uk
```

The recommended ssh config is:

```
Host cvmfs
    HostName cvmfs-upload01.gridpp.rl.ac.uk
    User northgridsgm
    ControlPath ~/.ssh/control-%r@%h:%p
    ControlMaster auto
    ControlPersist yes
```

Then starts to write something in `~/cvmfs_repo/simonsobservatory`. What you write will immediately be available at `/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory` on this publishing node. But it will only be synchronized to other sites with a time scale of around 2.5-3 hours.

On `vm77`, check if you see your stuffs is in `/cvmfs/northgrid.gridpp.ac.uk/simonsobservatory` already, if so, you can start to submit jobs that reads from there.

7.1.3 Monitoring

`condor_status`

See condor_status — HTCondor Manual for details.

`sudo condor_status`
: list all nodes in the pool along with their basic status information

`sudo condor_status -long`
: for more detailed information about each node

`sudo condor_status -constraint 'Arch == "x86_64"'`
: see available nodes after constraints

`condor_status -avail`
: Lists available nodes

`sudo condor_status -format "%s\n" Machine | sort -u`
: Lists only the machine names

`sudo condor_status -autoformat Machine Arch Microarch | sort -u`
: Auto-format instead.

7.2 Installing this project

This project has a few major components:

`docs/`
: documentation of the SO:UK Data Centre with a focus on software deployment.

`src/souk/`
: a Python library under the namespace of `souk`, intended to be convenient utilities to assist interacting and using the SO:UK Data Centre resource. It is currently empty but is required to be installed in order for the documentation to be built successfully (because API doc is built automatically).

7.2.1 Installing the environment

Using `pip`

```
python -m pip install .
# or if you want to install in editable mode
python -m pip install -e .
```

Using `conda`/`mamba` (recommended)

This is the method used to build the documentation here using Continuous Integration such as GitHub Pages and Read the Docs.

1. If you haven't already, install `conda` or `mamba` following your favorite guide.

   A one-liner to install `mamba` to `"$HOME/.mambaforge"` is provided:

   ```
   curl -L https://github.com/ickc/bootstrapping-os-environments/raw/master/install/
   ↪mamba.sh | bash
   ```

2. Install the environment

   ```
   mamba env create -f environment.yml
   # or using conda
   conda env create -f environment.yml
   ```

3. Activating the environment

   ```
   conda activate soukdc
   ```

4. Install this project

   ```
   python -m pip install --no-dependencies .
   # or if you want to install in editable mode
   python -m pip install --no-dependencies -e .
   ```

Note: This is exactly how the environment is prepared in GitHub Pages. See the source of `.github/workflows/sphinx.yml`.

7.3 Writing & building documentation

The main framework of choice to write and build documentation in this project is the Sphinx documentation generator, with the contents authored in markdown with the parser MyST. As RST is natively supported in Sphinx, you can also authored in RST.

7.3.1 Syntax

Markdown syntax are generally supported. One important note is how to include new files in the TOC tree. For example, in `maintainer.md`, the following `toctree` directive is used to include this file.

````
```{toctree}
:maxdepth: 2
:hidden:

maintainer/installing
maintainer/documenting
maintainer/releasing
```
````

If a new file is not included in a `toctree`, you will see a warning when building the documentation. A glob pattern can be used to include files implicitly, such as

```
```{toctree}
:maxdepth: 2
:hidden:
:glob:

pipeline/*
```
```

7.3.2 Structure

The following is the tree structure as of writing. It borrows a subpage concept from Notion. For example, `maintainer.md` has an accompanying `maintainer/` directory, which includes some more source files such as `documenting.md`. This indicates that `documenting.md` "belongs" to (or is a subpage of) `maintainer.md`.

```
docs
├── changelog.md
├── conf.py
├── developer
│   ├── htcondor-glossary.md
│   ├── tips
│   │   ├── monitor.md
│   │   └── tail.md
│   └── tips.md
├── developer.md
├── index.md
├── maintainer
│   ├── computing-resources.md
│   ├── documenting.md
│   ├── installing.md
│   └── releasing.md
├── maintainer.md
├── user
│   ├── onboarding.md
│   ├── pipeline
│   │   ├── 1-classad
│   │   │   ├── 1-classad-interactive
│   │   │   │   ├── example.ini
│   │   │   │   └── makefile
│   │   │   ├── 1-classad-interactive.md
│   │   │   ├── 2-classad-vanilla
│   │   │   │   ├── example.ini
│   │   │   │   └── makefile
│   │   │   ├── 2-classad-vanilla.md
│   │   │   ├── 3-classad-parallel
│   │   │   │   ├── example.ini
│   │   │   │   └── makefile
│   │   │   ├── 3-classad-parallel.md
│   │   │   ├── 4-classad-transfer-files
│   │   │   │   ├── makefile
│   │   │   │   ├── repl.ini
│   │   │   │   └── repl.sh
│   │   │   ├── 4-classad-transfer-files-2
│   │   │   │   ├── cat.ini
│   │   │   │   ├── cat.txt
│   │   │   │   └── makefile
```

(continues on next page)

(continued from previous page)

```
│   │   │   └── 4-classad-transfer-files.md
│   │   ├── 1-classad.md
│   │   ├── 2-software-deployment
│   │   │   ├── 1-tarball
│   │   │   │   ├── makefile
│   │   │   │   ├── tarball.ini
│   │   │   │   └── tarball.sh
│   │   │   ├── 1-tarball.md
│   │   │   └── 2-CVMFS.md
│   │   ├── 2-software-deployment.md
│   │   ├── 3-MPI-applications
│   │   │   ├── 1-OpenMPI
│   │   │   │   ├── env.sh
│   │   │   │   ├── makefile
│   │   │   │   ├── mpi.ini
│   │   │   │   └── mpi.sh
│   │   │   └── 1-OpenMPI.md
│   │   ├── 3-MPI-applications.md
│   │   ├── 4-IO
│   │   │   ├── 0-transfer-via-ClassAd.md
│   │   │   ├── 1-grid-storage-system.md
│   │   │   └── 2-librarian.md
│   │   └── 4-IO.md
│   ├── pipeline.md
│   └── quick-start.md
└── user.md

19 directories, 53 files
```

7.3.3 More details

See `pyproject.toml` or `environment.yml` to see the dependencies in Python. See `docs/conf.py` to see the extensions enabled in Sphinx.

7.4 Making a new release

7.4.1 Building

For the documentation, we aim to deliver it in these formats: HTML, ePub, PDF.

Per commit in `main` branch, if it builds successfully, it will deploy to the `latest` version in Read the Doc: https://docs.souk.ac.uk/en/latest/, and also in GitHub Pages: https://docs-ci.souk.ac.uk/.

To build the project, `makefile` is used. `make doc` (after you activated the `soukdc` conda environment) should builds the HTML documentation.

Warning: To ensure the documentation deploys to Read the Doc successfully, make sure there is no warnings when making the documentation. To ensure the cache isn't hiding some errors, you may run

```
make clean && make all
```

All targets (HTML, ePub, PDF) is deployed to Read the Doc automatically.

7.4.2 Serving

When authoring the documentation, you may want to have the HTML built at real time. Use

```
# after activating your conda environment soukdc
make serve
```

7.4.3 Semantic versioning and `bump-my-version`

Semantic versioning is followed, with the `MAJOR.MINOR.PATCH` convention as usual. Version `0.x.x` indicates the SO:UK Data Centre is not in final production ready state yet. `MINOR` version is bumped only if there's major functional improvement to the user experience, for example, when CVMFS is deployed. Otherwise, it is always a `PATCH` version release.

> **Warning:** Before making a new release, check Read the Doc in https://readthedocs.org/projects/souk-data-centre/builds/ to see if the `latest` build is successful first.

`bump-my-version` is used to automatically bump the version string scattered in multiple files. See `pyproject.toml` under `[tool.bumpversion]` for details.

To make a new release,

1. Update the `docs/changelog.md` to includes changes made since last release. GitHub can be useful here: https://github.com/simonsobs-uk/data-centre/compare/v0.4.1...main.
2. Make sure there's no uncommitted changes.
3. Run
   - `make bump PART=patch` for patch release,
   - `make bump PART=minor` for minor release,
   - `make bump PART=major` for major release.
4. Check Read the Doc at https://readthedocs.org/projects/souk-data-centre/builds/ to see the new builds are deployed successfully.
5. Make a new GitHub Releases based on the new tag created in step 3: https://github.com/simonsobs-uk/data-centre/releases/new?tag=v0.4.1.
   - title: SO:UK Data Centre 0.4.1 Documentation
   - content: copied from the changelog you updated in step 1.
   - check "Create a discussion for this release"
6. (Optional) make announcement in simonsobs-uk/data-centre · Discussions · GitHub. Follow the example in https://github.com/simonsobs-uk/data-centre/discussions/48#discussioncomment-8763151. Maintainer need to keep a list of GitHub handles elsewhere. Hint: maintain Kolen's Notion databases.

7.4.4 Releasing to `/opt/simonsobservatory` on `vm77`

`make opt`.

7.4.5 Single file targets

`make man txt` and upload to https://github.com/simonsobs-uk/data-centre/releases/latest manually.

TODO: automate this.

This is intended to be read by maintainers who typically has elevated privilege in the data centre system and/or this repository.

CHAPTER

EIGHT

PRESENTATIONS

8.1 2023-11-13 SO:UK BB day—Introduction to the SO:UK Data Centre

Note: Initially presented on 2023-11-13 shortly after the v0.2.0 release of the SO:UK Data Centre documentation. See *Changelog* to see what's changed since then.

8.1.1 Big picture (Why)

HPC vs. HTC

According to the European Grid Infrastructure (EGI):

High Throughput Computing (HTC)
: A computing paradigm that focuses on the efficient execution of a large number of **loosely-coupled tasks**. Given the **minimal parallel communication requirements**, the tasks can be executed on clusters or physically distributed resources using grid technologies. HTC systems are typically optimised to maximise the throughput over a long period of time and a typical metric is jobs per month or year.

High Performance Computing (HPC)
: A computing paradigm that focuses on the efficient execution of **compute intensive, tightly-coupled tasks**. Given the high parallel communication requirements, the tasks are typically executed on **low latency interconnects** which makes it possible to **share data very rapidly** between a large numbers of processors working on the same problem. HPC systems are delivered through low latency clusters and supercomputers and are typically optimised to maximise the number of operations per seconds. The typical metrics are FLOPS, tasks/s, I/O rates.

- High Throughput Computing - EGI Glossary - EGI Confluence
- High Performance Computing - EGI Glossary - EGI Confluence

	HTC	HPC
Optimized for	loosely-coupled tasks	tightly-coupled tasks
interconnects	high latency, low bandwidth (e.g. 10Gbps)	low latency, high bandwidth (e.g. InfiniBand)
Computational capability	subset of HPC	superset of HTC
Costs	lower costs per node, hence higher throughput per system budget	more expensive interconnects, high performance storage systems, specialized kernels
Parallelism	While technically possible, does not scale well beyond 1 node	Massively parallel, capable of using full machine for a single task
Homogeneity	Very forgiving in heterogeneous nodes	Highly homogeneous nodes, but increasing becomes heterogeneous within node (GPGPU)
MPI support	MPI support is an afterthought	MPI support is first class, MPI applications dominate
a.k.a.	The grid, grid computing (technically a subset of HTCs)	Supercomputers (technically a subset of HPCs)
Loved by	HEPs (CERN)	Everyone

Examples of HPC & HTC in CMB data analysis

To over-simplify, the amount of memory needed in a scientific application dictates which kind of computational resources are more apt.

- Cosmoglobe style full Bayesian analysis on CMB data: all data is needed in memory. E.g. full Perlmutter system probably would be barely (in)capable for doing full Bayesian analysis on SO data
- maximal likelihood / Madam (unbiased) mapmaking: all (partial if assumptions made) data per observatory frequency is needed in memory. E.g. Planck, SO LATs, etc.
- Naïve (filter/bin, biased) mapmaking: In principle, you only need to have enough memory for a subscan.

The NERSC problem

- See "[Mp107-u] NERSC use in 2023" email sent on Friday, Jan 20 2023 at 7:15 PM. My summary of this situation:
  - Typically the NERSC Allocation Request Form (ERCAP) is needed for a formal requests to utilize NERSC in computational research
  - CMB community is spoiled for 25 years that the application is done for all of us behind the scene, and has never been a shortage of supply (approved allocation)
  - all users are granted a generous amount of NERSC hours, based only on self-discipline
  - "a horse that harms the herd" (□□□□) from within the CMB community using NERSC resources irresponsibly
- In the 2023 Allocation Year, 0.1% of whole NERSC is allocated to SO, equivalent to a full ~ 0.1 PFLOPS machine.
- By my estimation SO:UK Data Centre would be of a similar order of magnitude (~ 0.1 PFLOPS), where whole cluster at Blackett is about 10 times as much.

HTC for SATs

- filter-bin (naïve mapmaking) mapmaking is suitable to be written in the map-reduce paradigm
- Riedel et al. (2019)[1] first demonstrated adapting the filter-bin mapmaking pipelines to utilize the Open Science Grid (OSG)
- SO:UK Data Centre
  - is funded to perform SO SATs analysis,
  - located from within Blackett, a grid similar to OSG, which is an HTC,
  - funded for 8 years, a stable long term commitment to the Science Readiness of SO SATs analysis.

8.1.2 Soft-launching SO:UK Data Centre v0.2.0 (What)

What is the SO:UK Data Centre

- Physically and infrastructurally, it is located within Blackett.
- Amounts to 10% of Blackett in terms of available CPUs. Have access to most of the available nodes.
- In the way of interacting with the computational resources, we are unique in the sense that Blackett users so far are submitting their jobs very differently through DiracUI within logging into a login / submit node. We however will be logging in and use HTCondor directly.
- HTCondor itself, while an inferior job manager comparing to SLURM in handling massively parallel applications, can be viewed as SLURM-like in many aspects. However, other design choices in Blackett contributes to other differences from NERSC for example.

Live demo

- SO:UK Data Centre 0.2.0 documentation
  - 1.1. Onboarding - SO:UK Data Centre 0.2.0 documentation
  - 1.2. Quick start - SO:UK Data Centre 0.2.0 documentation
  - 1.3. Running pipelines - SO:UK Data Centre 0.2.0 documentation
    - 1.3.3.1. OpenMPI - SO:UK Data Centre 0.2.0 documentation
    - 1.3.4.2. The grid storage system - SO:UK Data Centre 0.2.0 documentation

[1] Riedel, Benedikt, Lincoln Bryant, John Carlstrom, Thomas Crawford, Robert W. Gardner, Nicholas Harrington, Nicholas Huang, Alexandra Rahlin, Judith Stephen, and Nathan Whitehorn. 2019. "SPT-3G Computing." Edited by A. Forti, L. Betev, M. Litmaath, O. Smirnova, and P. Hristov. EPJ Web of Conferences 214: 03051. https://doi.org/10.1051/epjconf/201921403051.

8.1.3 To be explored (How)

How to design a workflow for a system

- To run a workflow effectively, it needs to be tailored for a system. Effectiveness can be measured in
  1. minimizing the amount of node-hours used (minimizing cost of "fair-share"),
  2. shortening the turn-around time.
- For a more capable system, it is more lenient on sub-optimality. I.e. adapting a workflow that works well at NERSC can be challenging at SO:UK Data Centre.
  - Even when a workflow works at NERSC, tailoring it for that system would makes it more "effective" as defined above.
  - E.g. for filter-bin mapmaking, you could run a big MPI job where each process is not communicating with each other.
    - This is wasting the capability of NERSC's HPC capability however.
    - It also creates more issues, such as load-balancing, that can be delegated to the job manager (scheduler) instead.
  - For NERSC, Job Arrays is a better way to launch such jobs, minimizing node-hours (because of load-balancing) and shortening the turn-around time (as the scheduler can allocate smaller jobs fitting into "cracks" earlier). This also maximize the utilization of NERSC, a cost hidden from your allocation.
- Recommended workflow: each submitted job should be "atomic", in the sense that it is the smallest piece of independent job that requires some sort of coordination within such job.
  - That means there's a lot of small jobs ($O(10,000)$ or more) you need to submit, a workflow manager is needed.
  - Workflow managers are independent of, often cooperate with, job managers. Example job managers are SLURM at NERSC, HTCondor in SO:UK Data Centre. Example workflow managers would be make (GNU make, makefile), snakemake, GNU Parallel, Parsl, Nextflow, of DAGMan in HTCondor.
  - Roll-your-own workflow manager are discouraged. Complexities of workflow managers:
    - Is it job-manager agnostics? Is the presence SLURM implicitly assumed?
    - How job dependencies are handled? Is the dependency graphs automatically generated and jobs submitted? E.g. after maps are made, how the next pipeline with MASTER is launched?
    - How failed jobs are handled? Do you need to keep track of corrupted output files (e.g. due to exceeding requested wall-clock time, memory available on node, etc.)? Will failed jobs be relaunched automatically?
- Caveats: while filter-bin mapmaking is well-suited for MapReduce paradigm, be careful will how you treat the data from each "map". Does it write the output to files and have the next pipeline reading it from the disks again? Beware of the explosion in intermediate disk space needed, as well as the congestion in interconnects either explicitly or implicitly. This is going to be important in co-addition of maps as well as null-split of maps where if handled not carefully would leads to data explosions and interconnect congestions.

Documentation

Documentation available at SO:UK Data Centre 0.2.0 documentation. How to use

- Search
- Different formats are available,
  - PDF / ePub / single page HTML from ReadTheDocs
  - man page and plain text available from GitHub Releases
- Single file output are well-suited to Chat with LLMs:
  - ChatGPT
  - Claude
- Collaborate and discuss on GitHub: simonsobs-uk/data-centre: This tracks the issues in the baseline design of the SO:UK Data Centre at Blackett

---

This section includes various presentations on SO:UK Data Centre which serves as an alternative medium to consume knowledge about our Data Centre. These are frozen in time however, so stick to the documentation for the latest information.

CHAPTER

NINE

SOUK

9.1 souk package

9.1.1 Subpackages

souk.util package

Subpackages

souk.util.cli package

Submodules

souk.util.cli.arg_string module

Prints a string with the given arguments.

Usage:

arg_string path/to/template.txt arg1=value1 arg2=value2

souk.util.cli.arg_string.**arg_string**(*path: Path*, ***kwargs: str*) → None

Prints a string with the given arguments.

souk.util.cli.arg_string.**cli**() → None

souk.util.cli.ini_formatter module

souk.util.cli.ini_formatter.**cli**() → None

souk.util.cli.ini_formatter.**ini_formatter**(*path: Path*, ***, *align_column: bool = False*, *sort: bool = False*) → None

Format INI file inplace.

param: path: Path to INI file. param: align_column: Align column. param: sort: Sort keys

souk.util.cli.ini_formatter.**main**(*glob_pattern: str*, ***, *align_column: bool = False*, *sort: bool = False*) → None

Format INI file inplace.

param: glob_pattern: Glob pattern to match files. param: align_column: Align column. param: sort: Sort keys

9.1.2 Submodules

souk.htcondor_helper module

souk.htcondor_helper.**get_hostnames**() → list[str]

Get hostnames of all machines.

Similar to this command:

run sudo condor_status -format "%sn" Machine | sort -u

CHAPTER

TEN

CHANGELOG

- v0.4.1: Minor improvements
  - doc
    - for user
      - fix link
    - for maintainer
      - add recommended CVMFS ssh config for maintainers
      - cleanup instruction on making new releases
- v0.4.0: Miscellaneous improvements
  - doc:
    - additions:
      - “Tips and gotchas when writing HTCondor ClassAds”
      - “Killing your jobs”
      - “Modifying our maintained software environments for software development”
      - “System specifications”
    - improvements:
      - on adding new users by maintainers
      - for new users to get onboarding
      - for new users to ssh for the first time
      - on launch MPI job in the vanilla universe (set `*_NUM_THREADS` explicitly)
    - use our own domain name at https://docs.souk.ac.uk
    - update badges
  - library:
    - add ini_formatter
    - add arg_string
    - improve arch_info
  - technical:
    - add CI for linkcheck

        * remove YAML safe dump
        * improve ClassAd white spaces in documentation
        * add formatters for py, ini
            · setup mypy and fix mypy errors
- v0.3.2: Fix PyPI README
- v0.3.1: First PyPI release
  - Python package:
    * rename the Python package to souk and publish to PyPI
    * add `arch_info`
  - Doc:
    * add external links
    * update maintainer guide to add new users
    * CVMFS
      · add "General softwares available from CVMFS"
      · add "Introduction to CVMFS"
    * add "The upcoming upgrade to Ceph & CephFS"
    * refactor section for interactive use, anticipating further expansion on this section
- v0.3.0: Supporting CVMFS
  - Deprecation
    * `/opt/simonsobservatory/pmpm-20230718-Linux-x86_64-OpenMPI.tar.gz` on `vm77` is deprecated. It is scheduled to be removed when `v0.4.0` is released, due in mid-Jan. 2024.
  - doc:
    * refactor quick-start guide w/ CVMFS & grid-storage
      · document how to use CVMFS to load softwares (software deployment)
      · write about continuous deployment
      · refactor "Grid Storage Systems" across user guide and developer guide, to focus only on the simple, essential stuffs in the user guide.
    * simplify how AC is setup on worker nodes
    * add an example to run MPI applications in Vanilla universe
    * document the behavior of `/tmp` on worker nodes
    * add presentations
    * add guide to log-in, including ssh-config.
    * document wall-clock time
    * document Apptainer
    * add basic documentation on systems
    * add badges
    * improve readme / intro with relevant links.

  - for maintainers
    - add guide to register as Content Manager
      - add guide to publish to CVMFS
    - maintainer's guide add monitoring
  - technical:
    - fix GitHub Pages
      - deployment path
      - fetch-depth
    - Release all available formats on Read the Docs
    - improve `console` in syntax highlighting
    - improve build systems with more targets
    - upgrade bump2version to bump-my-version
- v0.2.0: Supporting the grid storage system
  - doc:
    - add sections on the grid storage systems and refactor the IO sections
    - add `docs/maintainer/computing-resources.md`
    - add `docs/user/onboarding.md`
  - technical:
    - migrate GitHub Pages to use GitHub Actions
- v0.1.2: Start writing maintainer guide
  - documentation enhancements and bug fixes:
    - add maintainer sections
  - misc. software related:
    - fix discrepancies between `environment.yml` and `pyproject.toml`
    - fix `__version__` in `souk`
- v0.1.1: Start writing developer guide
  - documentation enhancements and bug fixes:
    - add sections on tips, including how to monitor jobs and the use of `tail` to stream real time std-out/stderr.
    - document `cat.txt` in `4-classad-transfer-files.md`
  - misc. software related:
    - use bump2version
    - fix `WARNING: more than one target found for 'myst' cross-reference monitor: could be :std:ref:`Monitor your jobs `or :std:doc:`Monitor your jobs `[myst.xref_ambiguous]`
- v0.1.0: Initial release to a small number of testers from SO.

CHAPTER

ELEVEN

EXTERNAL LINKS

Facilities:

- Services – The Blackett Computing Facility
- NERSC Documentation

Grid computing:

- VOMS Documentation
- VOMS Admin Server
- HTCondor Manual
- GFAL2 · Data Management Clients Documentation
- Welcome to CernVM-FS's documentation!
- CVMFS - GridPP Wiki

Documentation:

- Welcome — Sphinx documentation
- MyST Overview
- Semantic Versioning

CHAPTER

TWELVE

SO:UK DATA CENTRE

This documents the baseline design of the SO:UK Data Centre at Blackett.

The GitHub repository simonsobs-uk/data-centre contains

- source of the documentation, including the codes in the documentation that you can run directly,
- the Issues tracker as well as Discussions, for any data centre related bugs or questions, and
- a Python package for data centre maintenance.

To access our documentation, you have a few options, (in the order from convenience to advanced usages):

1. SO:UK Data Centre documentation at Read the Docs. This gives you access to all versions of SO:UK Data Centre documentations, as well as multiple output formats including HTML, ePub, PDF.
2. SO:UK Data Centre documentation at GitHub Pages which is HTML only and points to the latest commits only.
3. SO:UK Data Centre documentation at GitHub Releases. This gives you additional output formats such as man page and plain text.

Note that Read the Docs serves the searches from server-side powered by Elasticsearch. So searches from Read the Docs and GitHub Pages will gives you different results. Try the Read the Docs first for better results and fall back to GitHub Pages.

Lastly, those single file documentation formats are very suitable for feeding into Large Language Models (LLMs). For example, try downloading our plain text format and upload it to ChatGPT or Claude and start chatting. You can ask them to explain things in the documentations in details that we cannot covers here.

PYTHON MODULE INDEX

S

INDEX