What is the Open Science Grid?

The OSG consortium of research collaborations, campuses, national laboratories, and software providers is dedicated to the advancement of all open science via the practice of distributed High Throughput Computing (dHTC), and to the advancement of its state of the art.

What is the Open Science Pool?

The OSPool provides researchers with **fair-share** access to computing and data capacity powered by distributed high-throughput computing (dHTC) technologies.

- Open for any scientist or group, of all disciplines, doing open science in the US.
- Built to run independent computations on a massive scale.
- The OSPool is built from resources contributed by university campuses, government-supported supercomputing facilities and research collaborations
- Resources are assigned on a fair-share basis at no cost to researchers; no allocation process is required.

Recommended Video Guide: https://www.youtube.com/watch?v=9896xAhT4dY

Video discusses:

- HTCondor History and Status
- Terminology
- Job Matching & Execution
- **Basic Submit File**
- Job Idle, Running & Completes

Additional Guides:

Introduction to HTC Condor:

- What is HTCondor?
- Running a Job with HTCondor
- How HTCondor Matches and Runs Jobs
- Submitting Multiple Jobs with HTCondor
- Testing and Troubleshooting
- Use Cases and HTCondor Features

HTC Manual on Submitting Jobs

OSG Basic Getting Started: Basic Exercise – Simple Submit Job

Choosing a Universe – HTC Condor

Submitting a Job:

The condor_submit command takes a job description file as input and submits the job to HTCondor. In the submit description file, HTCondor finds everything it needs to know about the job. Items such as the name of the executable to run, the initial working directory, and command-line arguments to the program all go into the submit description file. condor_submit creates a job ClassAd based upon the information, and HTCondor works toward running the job.

It is easy to submit multiple runs of a program to HTCondor with a single submit description file. To run the same program many times with different input data sets, arrange the data files accordingly so that each run reads its own input, and each run writes its own output.

Sample Submission Files:

Simplest Submission File:

It queues the program myexe for execution somewhere in the pool. As this submit description file does not request a specific operating system to run on, HTCondor will use the default, which is to run the job on a machine which has the same architecture and operating system it was submitted from.

Before submitting a job to HTCondor, it is a good idea to test it first locally, by running it from a command shell.

$ ./myexe SomeArgument

```
# Example 1
# Simple HTCondor submit description file
# Everything with a leading # is a comment

executable   = myexe
arguments    = SomeArgument

output       = out/job$(ProcId).out
error        = err/job$(ProcId).err
log          = log/job$(ProcId).log

request_cpus   = 1
request_memory = 1024
request_disk   = 10240

should_transfer_files = yes
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

queue 1
```

- **Executable**: The script or command you want HTCondor to run

- **Arguments**: Include any arguments required as parameters for your program. When your program is executed, the Condor software issues the string assigned to this attribute as a command-line argument
    - Any arguments that could be passed to the command. – Command Line Args

- **Output**: Where the STDOUT of the command or script should be written to.
- **Error:** This schedules the job. It becomes more important (along with the interpolation) when queue is used to schedule multiple jobs by taking an integer as a value.
- **Log**: This is the output of HTCondor's logs for your jobs, not any logging your job itself will perform. It will show the submission times, execution host and times, and on termination will show stats.

- **Request the appropriate resources for your job to run**
    - **Request_CPU:** For requesting CPU cores start by requesting a single cpu. With single-cpu jobs, you will see your jobs start sooner. Ultimately you will be able to achieve greater throughput with single cpus jobs compared to jobs that request and use multiple cpus.
        - Keep in mind, requesting more CPU cores for a job does not mean that your jobs will use more cpus. Rather, you want to make sure that your CPU request matches the number of cores (i.e. 'threads' or 'processes') that you expect your software to use. (Most softwares only use 1 CPU core, by default.)
    - **Request_Memory:** Estimating memory requests can sometimes be tricky. If you've performed the same or similar work on another computer, consider using the amount of memory (i.e. RAM) from that computer as a starting point. For instance, most laptop computers these days will have 8 or 16 GB of memory, which is okay to start with if you know a single job will succeed on your laptop.
        - For your initial tests it is OK to request more memory than your job may need so that the test completes successfully. The key is to adjust memory requests for subsequent jobs based on the results of these test jobs.
    - **Request_Disk**: To inform initial disk requests always look at the size of your input files. At a minimum, you need to request enough disk to support all of the input files, executable, and the output you expect, but don't forget that the standard 'error' and 'output' files you specify will capture 'terminal' output that may add up, too.

    - **Should_Transfer_Files:** Setting the should_transfer_files command explicitly enables or disables the file transfer mechanism. The command takes on one of three possible values:
        - YES: HTCondor transfers both the executable and the file defined by the input command from the machine where the job is submitted to the remote machine where the job is to be executed. The file defined by the output command as well as any files created by the execution of the job is transferred back to the machine where the job was submitted. When files are transferred and the directory location of the files is determined by the command when_to_transfer_output.
        - IF_NEEDED: HTCondor transfers files if the job is matched with and to be executed on a machine in a different FileSystemDomain than the one the submit machine belongs to, the same as if should_transfer_files = YES. If the job is matched with a machine in the local FileSystemDomain, HTCondor will not transfer files and relies on the shared file system.

        - NO: HTCondor's file transfer mechanism is disabled.
    - **Transfer_Input_Files**: If the job requires other input files, the submit description file should utilize the transfer_input_files command. This comma-separated list specifies any other files or directories that HTCondor is to transfer to the remote scratch directory, to set up the execution environment for the job before it is run. These files are placed in the same directory as the job's executable.
        - If the file transfer mechanism is enabled, HTCondor will transfer the following files from the execute machine back to the submit machine after the job exits.
            - the output file, as defined with the output command
            - the error file, as defined with the error command
            - any files created by the job in the remote scratch directory; this only occurs for jobs other than grid universe, and for HTCondor-C grid universe jobs; directories created by the job within the remote scratch directory are ignored for this automatic detection of files to be transferred.
    - **When_To_Transfer_Output**: command tells HTCondor when output files are to be transferred back to the submit machine. The command takes on one of two possible values:
        - ON_EXIT: HTCondor transfers the file defined by the output command, as well as any other files in the remote scratch directory created by the job, back to the submit machine only when the job exits on its own.
        - ON_EXIT_OR_EVICT: HTCondor behaves the same as described for the value ON_EXIT when the job exits on its own. However, if, and each time the job is evicted from a machine, files are transferred back at eviction time. The files that are transferred back at eviction time may include intermediate files that are not part of the final output of the job. Before the job starts running again, all of the files that were stored when the job was last evicted are copied to the job's new remote scratch directory. The purpose of saving files at eviction time is to allow the job to resume from where it left off. This is similar to using the checkpoint feature of the standard universe, but just specifying ON_EXIT_OR_EVICT is not enough to make a job capable of producing or utilizing checkpoints. The job must be designed to save and restore its state using the files that are saved at eviction time.

- **Queue**: The command queue instructs the Condor system to submit one set of program, attributes, and input file for processing. You use this command one time for each input file that you choose to submit. (Note: this keyword should be specified without an equals sign, e.g. "Queue 10".)