



Scaling Out HTC Workloads

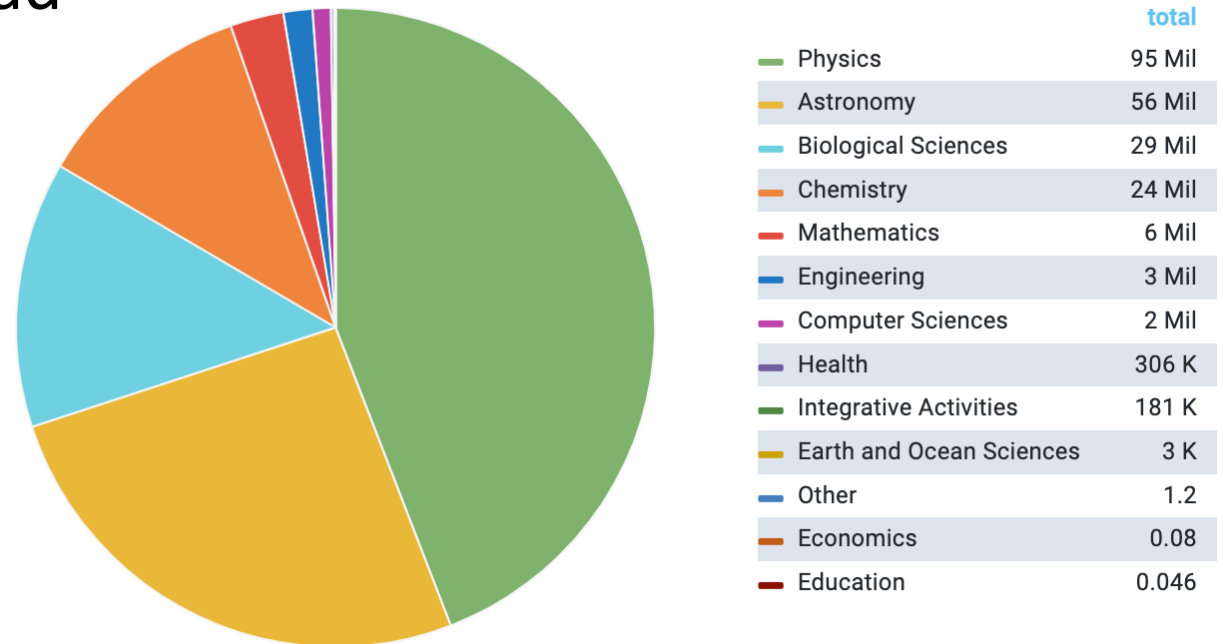
Rachel Lombardi

OSGUS 23

Agenda

- Why scale out with HTC?
- Steps for carrying out a HTC Workload
 - Assemble workload components
 - Organize workload components
 - Scaling Out
- Job Attributes
- Checking Completed Jobs
- Summary

Core Hours by Field of Science



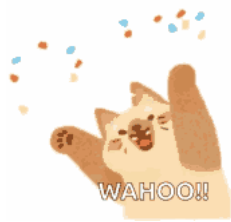


Why Scale Out with High Throughput Computing?

Why scale out with HTC?

Because we can *Divide and Conquer* to finish research faster, analyze more data, etc.

Basically, doing more
awesome research in
less time!





HTC Workload Steps



Part 0

Assemble Workload Components



Recap: HTC Workload Components

What components do you need for an HTC workload (or single job)?

We have talked about all of these things this week!

- Monday: HTCondor Job Submission
 - including coordinating input and output files
- Tuesday: Software
- Wednesday: Data

Reflection

What are the components you need for your HTC workload?

What results will you generate?

What other files will be generated by HTCondor or the jobs?





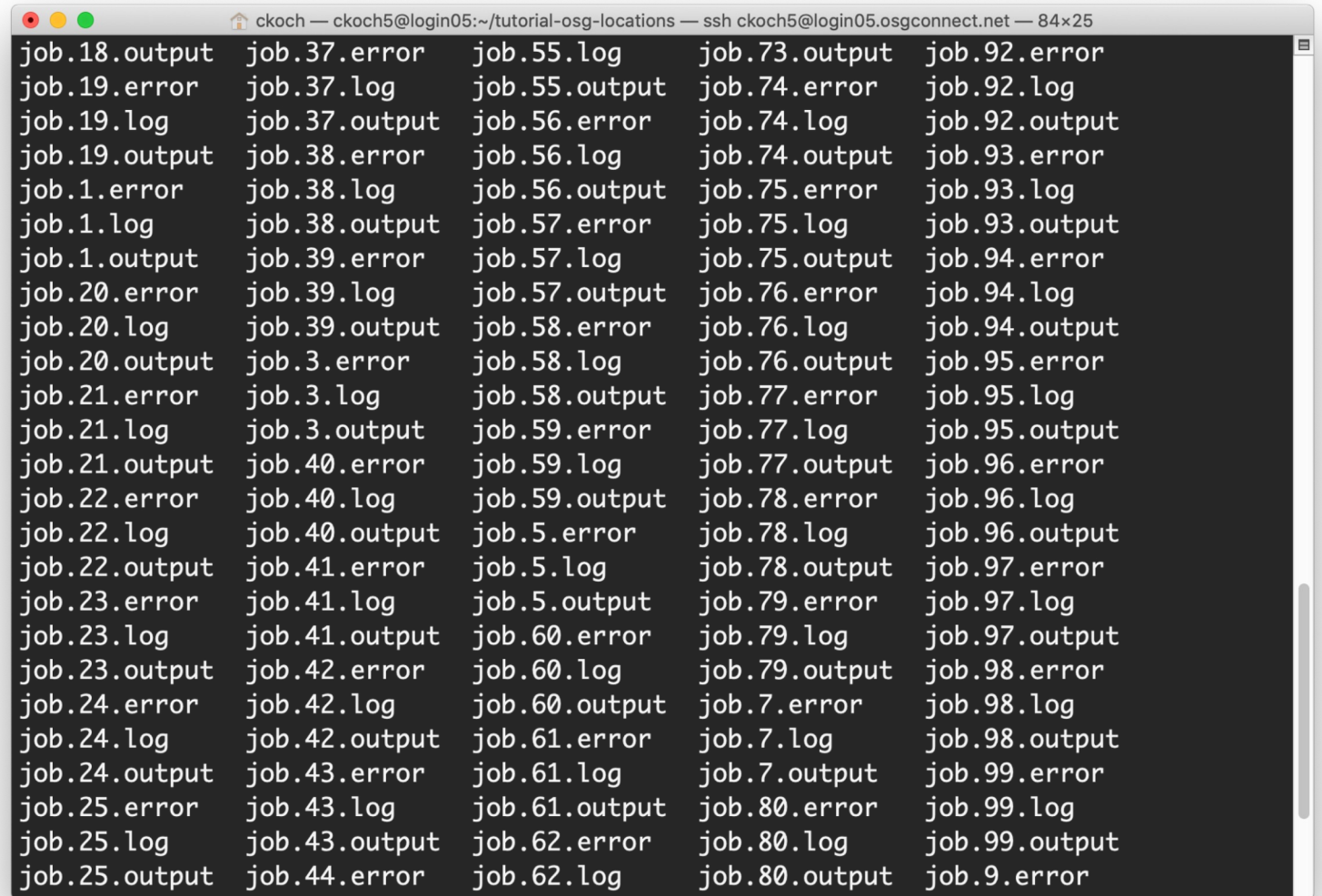
Part I

Organizing HTC Workload Components

Why organize?

By default, HTCondor writes all job files (input, output, HTCondor logs, etc.) back to the same place, which means your home directory can look something like this:

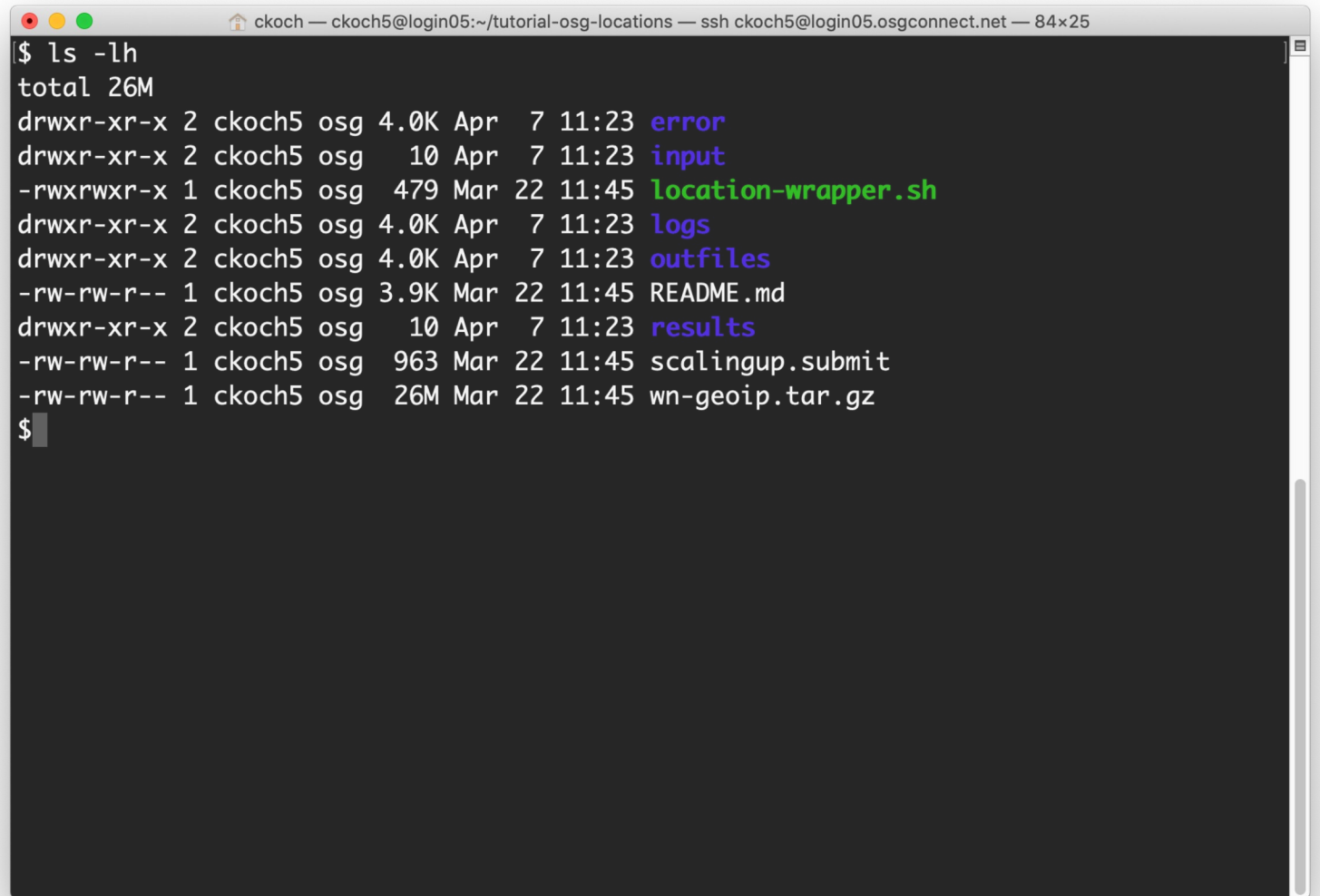
This makes it hard to find things!



```
ckoch — ckoch5@login05:~/tutorial-osg-locations — ssh ckoch5@login05.osgconnect.net — 84x25
job.18.output  job.37.error  job.55.log    job.73.output  job.92.error
job.19.error   job.37.log    job.55.output job.74.error   job.92.log
job.19.log     job.37.output job.56.error  job.74.log    job.92.output
job.19.output  job.38.error  job.56.log    job.74.output  job.93.error
job.1.error    job.38.log    job.56.output job.75.error   job.93.log
job.1.log      job.38.output job.57.error  job.75.log    job.93.output
job.1.output   job.39.error  job.57.log    job.75.output  job.94.error
job.20.error   job.39.log    job.57.output job.76.error   job.94.log
job.20.log     job.39.output job.58.error  job.76.log    job.94.output
job.20.output  job.3.error   job.58.log    job.76.output  job.95.error
job.21.error   job.3.log     job.58.output job.77.error   job.95.log
job.21.log     job.3.output  job.59.error  job.77.log    job.95.output
job.21.output  job.40.error  job.59.log    job.77.output  job.96.error
job.22.error   job.40.log    job.59.output job.78.error   job.96.log
job.22.log     job.40.output job.5.error   job.78.log    job.96.output
job.22.output  job.41.error  job.5.log     job.78.output  job.97.error
job.23.error   job.41.log    job.5.output  job.79.error   job.97.log
job.23.log     job.41.output job.60.error  job.79.log    job.97.output
job.23.output  job.42.error  job.60.log    job.79.output  job.98.error
job.24.error   job.42.log    job.60.output job.7.error    job.98.log
job.24.log     job.42.output job.61.error  job.7.log     job.98.output
job.24.output  job.43.error  job.61.log    job.7.output  job.99.error
job.25.error   job.43.log    job.61.output job.80.error   job.99.log
job.25.log     job.43.output job.62.error  job.80.log    job.99.output
job.25.output  job.44.error  job.62.log    job.80.output  job.9.error
```

Why organize?

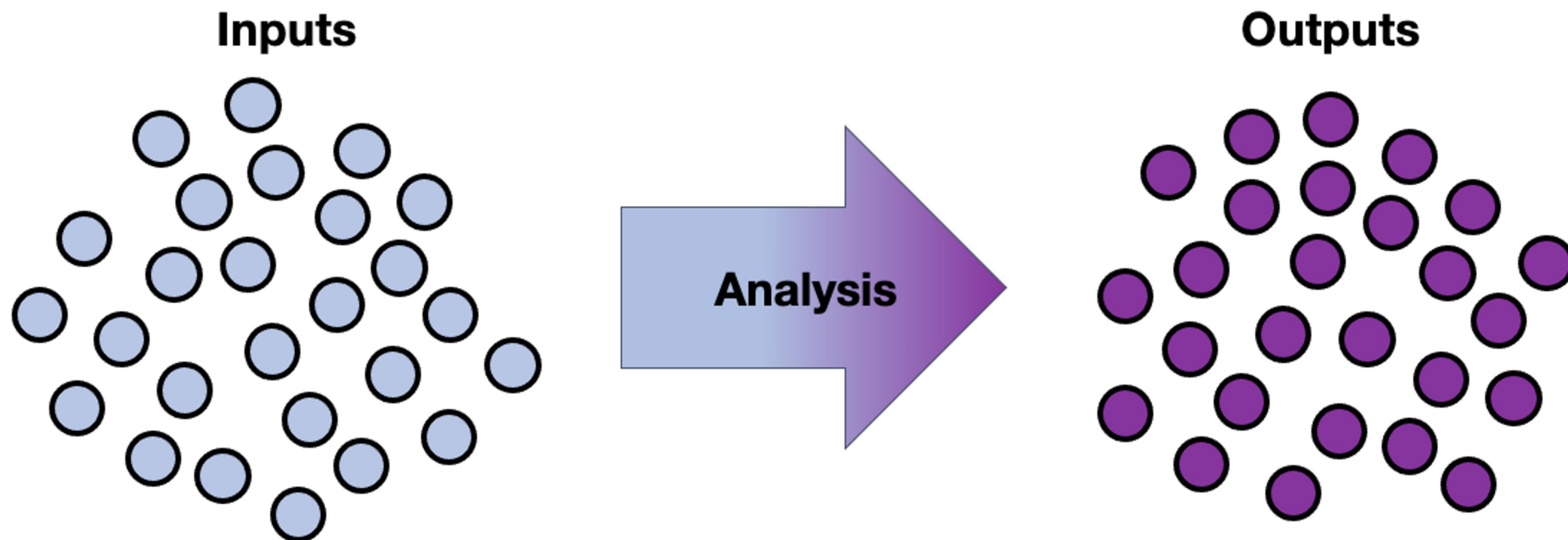
We can improve our workflow by intentionally organizing our input and output files on the Access Point.



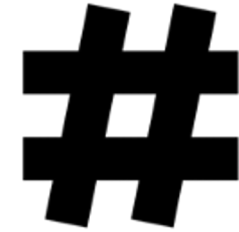
```
$ ls -lh
total 26M
drwxr-xr-x 2 ckoch5 osg 4.0K Apr  7 11:23 error
drwxr-xr-x 2 ckoch5 osg  10 Apr  7 11:23 input
-rwxrwxr-x 1 ckoch5 osg  479 Mar 22 11:45 location-wrapper.sh
drwxr-xr-x 2 ckoch5 osg 4.0K Apr  7 11:23 logs
drwxr-xr-x 2 ckoch5 osg 4.0K Apr  7 11:23 outfiles
-rw-rw-r-- 1 ckoch5 osg 3.9K Mar 22 11:45 README.md
drwxr-xr-x 2 ckoch5 osg  10 Apr  7 11:23 results
-rw-rw-r-- 1 ckoch5 osg  963 Mar 22 11:45 scalingup.submit
-rw-rw-r-- 1 ckoch5 osg 26M Mar 22 11:45 wn-geoip.tar.gz
$
```

HTC Workloads as Input/Output Sets

The next example will model workloads that use many input files to produce many output files.



Example: Text Analysis



Book text to analyze

Python script that counts the frequency of different words

Output counts of different words in book

```
$ ./wordcount.py Dracula.txt
```

Organizational Plan For Our Files

wordcount.sub

wordcount.py

input/

Dracula.txt

...

output/

count.Dracula.txt

...

We will assume that we want to put our input files (books) in one folder, and our output files (word counts) in another folder.

Organizational Plan For Our Files

```
wordcount.sub
wordcount.py
input/
    Dracula.txt
    ...
output/
    count.Dracula.txt
    ...
log/
    job.0.log
    ...
errout/
    job.0.out
    job.0.err
    ...
```

There are ***additional*** files that will be produced by the job as well that we should consider – the HTCondor log, stdout and stderr. We'll put these into two folders.

Coordinate HTCondor and File Structure

```
wordcount.sub
wordcount.py
input/
    Dracula.txt
    ...
output/
    count.Dracula.txt
    ...
log/
    job.0.log
    ...
errout/
    job.0.out
    job.0.err
    ...
```

```
# submit file name: wordcount.submit
executable = wordcount.py
arguments = Dracula.txt

transfer_input_files = inputs/Dracula.txt
transfer_output_remaps = \
    "count.Dracula.txt = outputs/count.Dracula.txt"

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

queue 1
```


HTCondor Options for Organizing Files

Syntax	Purpose	Features
<pre>Transfer_output_remaps = "file1.out=path/to/file1.out; file2.out=path/to/renamedFile2.out"</pre>	Used to save output files in a specific path and using a certain name	<ul style="list-style-type: none">- Used to save output files to a specific folder- Used to rename output files to avoid writing over existing files
<pre>Initialdir = path/to/initialDirectory</pre>	Sets the submission directory for each job. When set, this becomes the base path where output files will be saved.	<ul style="list-style-type: none">- Used to submit multiple jobs from different directories- Used to avoid having to write some paths in other submit file values

More Information: <https://htcondor.readthedocs.io/en/latest/users-manual/file-transfer.html>

Return Output to Specified Directory with InitialDir

```
submission_dir/  
  job.sub  
  exec.py  
  shared_vars.txt  
  results/  
    input.txt  
    output.txt  
    job.err  
    job.log  
    job.out
```

```
# File name: job.sub  
executable = exec.py  
  
initialdir = results  
transfer_input_files = input.txt,  
  ../shared_vars.txt  
  
log = job.log  
out = job.out  
error = job.err  
  
queue 1
```

Separate Jobs with InitialDir

```
submission_dir/
```

```
job.submit
```

```
analyze.exe
```

```
job0/
```

```
file.in job.log job.err  
file.out job.out
```

```
job1/
```

```
file.in job.log job.err  
file.out job.out
```

```
job2/
```

```
file.in job.log job.err  
file.out job.out
```

```
# File name = job.submit
```

```
executable = analyze.exe
```

```
initialdir = job$(ProcId)
```

```
arguments = file.in file.out
```

```
transfer_input_files = file.in
```

```
log = job.log
```

```
error = job.err
```

```
output = job.out
```

```
queue 3
```

Executable
should be in the
directory with the
submit file, *not*
in the individual
job directories



Organizing Data Files

Some HTC systems will have you place small files in one directory and larger files in a different directory.

For example, on ap40 (an OSPool Access Point), we place files less than 1GB in /home and larger files in an OSDF origin.

Once inputs and outputs are placed in the right location, use the appropriate HTCondor file transfer options to move the data to jobs.

Reflection

How big are the files in my input / output sets?

What organizational strategy makes sense for the next steps in my analysis?

- Do you want inputs in one folder and outputs in another folder? Use [transfer_output_remaps](#).
- Do you have many outputs for each job that you'd like to group together, but keep separate from other job outputs? Do you want to keep inputs/outputs for the same job together? Maybe use [initialdir](#).

How do you want to organize the HTCondor/system files?



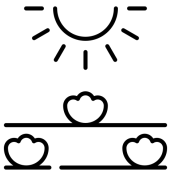


Part II Scaling Out

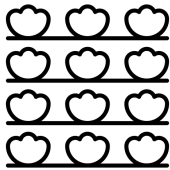
Overview of the Scaling Out Process



Created by Made from the Noun Project



Created by Made from the Noun Project



Created by Made from the Noun Project

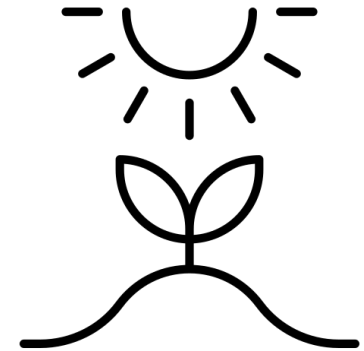
1. **Run a single job:** For each job type, get a test job working reliably & tune resource needs
2. **Test a small workload:** Scale up to ~10 jobs, checking reliability & Access Point resource demand
3. **Scale out:** Continue scaling up in 10–100× increments, checking for & fixing issues

Stage 1: Get One Job Running

You know how to do this! 😊

- Gather executable, inputs, arguments, etc.*
- Estimate initial resource needs*
- Write a submit file
- Submit!
- Review all outputs, including log, output, and error files
- Check actual resource usage and update resource needs*
- Repeat until (fairly) accurate and reliable

* More details on next slides



Created by Made
from the Noun Project

Stage 1: Tips for Initial Test Jobs

- Test one of each kind of job you will run (e.g., prep, simulation, analysis)
- Select smaller data sets or subsets of data for your first test jobs
- Pick test jobs that will reproduce results from elsewhere, if possible
- Name files carefully to help identify which results go with which tests
- Make sure you understand and can run your software
 - Software — executable, dependencies, maybe a wrapper script to prepare environment
 - Command-line arguments
 - Input files

Stage 1: Estimating Initial Resource Needs

CPU

- By default, start with 1
- Unless you know for sure that your executable uses a certain number > 1

Memory

- Start with the total memory available on laptop or where it ran before
- It's ok if this is a lot the first time, you will fine-tune later

Disk

- Estimate (as best you can) and sum sizes of: executable (+ environment), input files, output files, temporary files, standard output/error

Stage 1: Run, Refine, Repeat

After running a test job:

- Check logs and output for errors, warning, holds, etc.
- Check HTCondor job log for actual resource usage
- Fix issues, update resource needs, run 1 job again!
- Good opportunity to check “fit” of jobs to HTC and maybe adjust

```
005 (1234.000.000) 2022-07-28 09:12:34 Job terminated.
```

```
(1) Normal termination (return value 0)
```

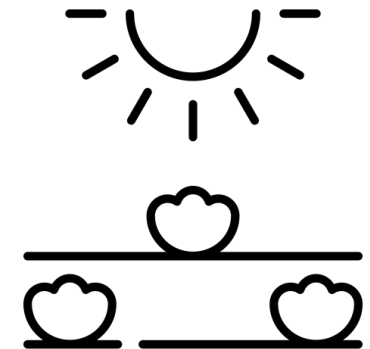
```
[...]
```

Partitionable Resources	:	Usage	Request	Allocated
Cpus	:		1	1
Disk (KB)	:	40	30	4203309
Memory (MB)	:	1	1	1

Stage 2: Scale To About 10 Jobs

For each kind of job, once you have 1 job working, try about 10

- Try a representative variety of arguments and input files
- Start developing methods for checking results of *all* jobs
- Estimate total resource needs for the Access Point itself
- **Repeat tests** at this scale until issues are fixed & resources are accurate



Created by Made
from the Noun Project

Stage 2: Try Various Inputs

For Stage 1, the suggestion was to keep things short and simple

- **For Stage 2, it is time to explore the entire range of inputs to your jobs**
 - Different command-line arguments; e.g., start, middle, and end of parameter sweep
 - Different input files; e.g., small, medium, and large — whatever makes sense for you
- As you explore, you may find that per-job resource needs vary
- Set your resource requests a bit higher than maximum observed usage
 - For example, if 10 test jobs used between 938 MB – 1.23 GB of memory, update your submit file to request 1.5 GB memory
- After any changes, run the same test again and re-evaluate

Stage 2: Checking Results of Multiple Jobs

Start developing methods for checking the results of multiple jobs

- Output from your executable (i.e., your research results)
- Debugging output: standard output and error files, executable logs, etc.
- HTCondor job log file (`log = job.log` in your submit file)

This may be one of the most overlooked aspects of scaling up!

- Checking 1 job is easy; checking 10 is tedious; checking 1000s by hand? 😭
- Techniques include:
 - Sampling
 - Developing tools to automate (see Part III)
- Sounds a bit like research, right? You know how to do that...

Stage 2: Estimate Access Point Needs

Do not forget about your Access Point – it is a shared resource, too!

- Storage space for files
 - Based on a run of 10 jobs, estimate total number and size of all files for full production
 - Do you have enough storage space on the Access Point? If not, what options exist?
- Number of running jobs
 - In theory, how many jobs could you have running at once?
 - Each running job uses some CPU and memory on the Access Point itself
 - If submitting over 10,000 jobs consider limiting (*throttling*) running and idle jobs on Access Point

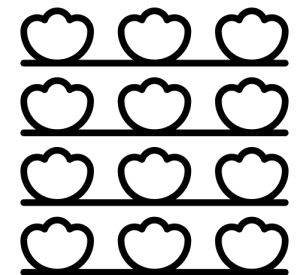
Stage 3: Iterate in Steps of 10–100×

By now, you have tested tens of jobs, maybe in a workflow; what next?

- Continue scaling up in increments of 10–100 times the number of jobs
- All the considerations from Stage 2 apply at each increment
- Be sure to find, understand, and hopefully fix issues before moving on

As you scale up, a challenge is to distinguish among:

- Real issues with your jobs, workflow, resource requests, etc.
- Real issues with certain subsets of your jobs
- Temporary issues with the HTC infrastructure itself
- Bugs and other longer-lasting issues with the infrastructure
- We can help! Email us with support requests if you get stuck.




Created by Made
from the Noun Project

Reflection

Where are you in the scaling up process?

What are three things you should consider in your current stage?





Part III

Tools for Monitoring



Tools for Learning About Jobs

HTCondor's job attribute information

- Accessed via `condor_q`, or `condor_history`

Files

- HTCondor log files
- Standard error/standard output files



Job Attributes

Job Attributes with condor_q

HTCondor stores a list of information about each job.

This information is stored in this format:

- **AttributeName** = value

You can find a list of attributes for a single job by running:

- `condor_q -l JobID`

You can print out specific attributes by using the “format” or “auto-format” flags with an HTCondor command:

- `condor_q -af Attribute1 Attribute2`
- adds job number: `condor_q -af:j Attribute1 Attribute2`

Interesting Job Attributes

Job identifying information

- ClusterID
- ProcID
- Cmd
- Arguments
- UserLog

Where it ran

- LastRemoteHost
- MATCH_EXP_JOBGLIDEIN_ResourceName

Resource Request and Usage

- RequestCpus (Memory, Disk)
- MemoryProvisioned (Disk)
- CPUsUsage (MemoryDisk)

Timing

- EnteredCurrentStatus
- QDate

Interesting Job Attributes

Codes

- JobStatus
- ExitCode
- HoldReasonCode
- HoldReasonSubCode,
- NumHoldsByReason

Counts

- NumJobStarts
- NumShadowStarts
- NumSystemHolds,



Checking Completed Jobs

Checking Completed Jobs

condor_history

- Contains finalized job attributes for completed jobs
 - some have different names (HoldReason --> LastHoldReason)
- Easy to use constrain and to display values (like condor_q)
- Can be slow to search (latest first) and may drop old records quickly

HTCondor job log files (log = *job.log* in submit)

- Contain a lot of information
- That is both a blessing and a curse
- Somewhat easy to parse — or use HTCondor Python bindings/other tools to help

HTCondor Job Log Files

- One big, combined file, or one per job? Your preference, really
- With tens or hundreds of jobs (& more), not practical to review manually
 - Can try to use the grep command-line tool to find specific lines
 - Use a tool to summarize log results (logs2csv)

HTCondor Job Log Files: Terminations

To find when every job ended:

```
$ grep '^005' LOGS
```

(*LOGS* can be one file, a list of files, or a glob (using ***) of files)

To find termination codes (exit codes) for every job:

```
$ grep termination LOGS
```

(will not show job IDs, though)

To get counts by termination code:

```
$ grep termination LOGS | sort | uniq -c
```

HTCondor Job Log Files: Resource Lines

To get memory resource lines:

```
$ grep -h 'Memory (MB) *:' LOGS > memory_resources.txt
```

To get disk resource lines:

```
$ grep -h 'Disk (KB) *:' LOGS > disk_resources.txt
```

Import the resulting files into Excel (with some attention to import options)

For file transfers:

```
$ grep -h 'Total Bytes Sent By Job' LOGS
```

```
$ grep -h 'Total Bytes Received By Job' LOGS
```

HTCondor Job Log Files: Checking on Holds

To view all job holds, their reasons, and related codes:

```
$ grep -h -A 2 '^012' LOGS
```

(Omit the -h option to see log filenames for each hit.)

Note: The OSPool may automatically release (rerun) some held jobs; if you don't look for them explicitly, you may never know those holds occurred

HTCSS Job Event Log to CSV Summarizer

A simple script that reads HTCondor job event logs and outputs a CSV summary of job statistics.

```
[alice@ap40 logs]$ logs2csv *.log > summary.csv
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	JobId	Submitted	ExecCount	EvictCount	TermCount	TermNorm	TermAbnorm	ShdwExcpt	Abrts	Suspnds	Unsus	Holds
2	15505	9/28/22 9:51	1	0	0	0	0	1	1	0	0	1
3	15507	9/28/22 10:05	1	0	0	0	0	1	1	0	0	1
4	15508	9/28/22 11:56	1	0	0	0	0	1	1	0	0	1
5	15509	9/28/22 15:22	1	0	0	0	0	1	1	0	0	1
6	15510	9/28/22 15:32	0	0	0	0	0	1	1	0	0	1
7	15511	9/28/22 16:46	1	0	0	0	0	1	1	0	0	1
8	15514	9/29/22 10:07	0	1	0	0	0	0	1	0	0	0
9	15515	9/29/22 10:09	1	0	0	0	0	1	1	0	0	1
10	15520	9/30/22 9:44	1	0	1	1	0	0	0	0	0	0
11	15651	10/27/22 13:12	1	0	1	1	0	0	0	0	0	0
12	15653	10/27/22 13:15	1	0	1	1	0	0	0	0	0	0
13	15654	10/27/22 13:16	1	0	1	1	0	0	0	0	0	0
14	282	6/28/22 12:20	1	0	1	1	0	0	0	0	0	0
15	283	6/29/22 9:22	1	0	0	0	0	1	1	0	0	1
16	284	6/29/22 9:30	1	0	0	0	0	1	1	0	0	1

<https://github.com/osg-htc/job-event-log-to-csv>



In Summary

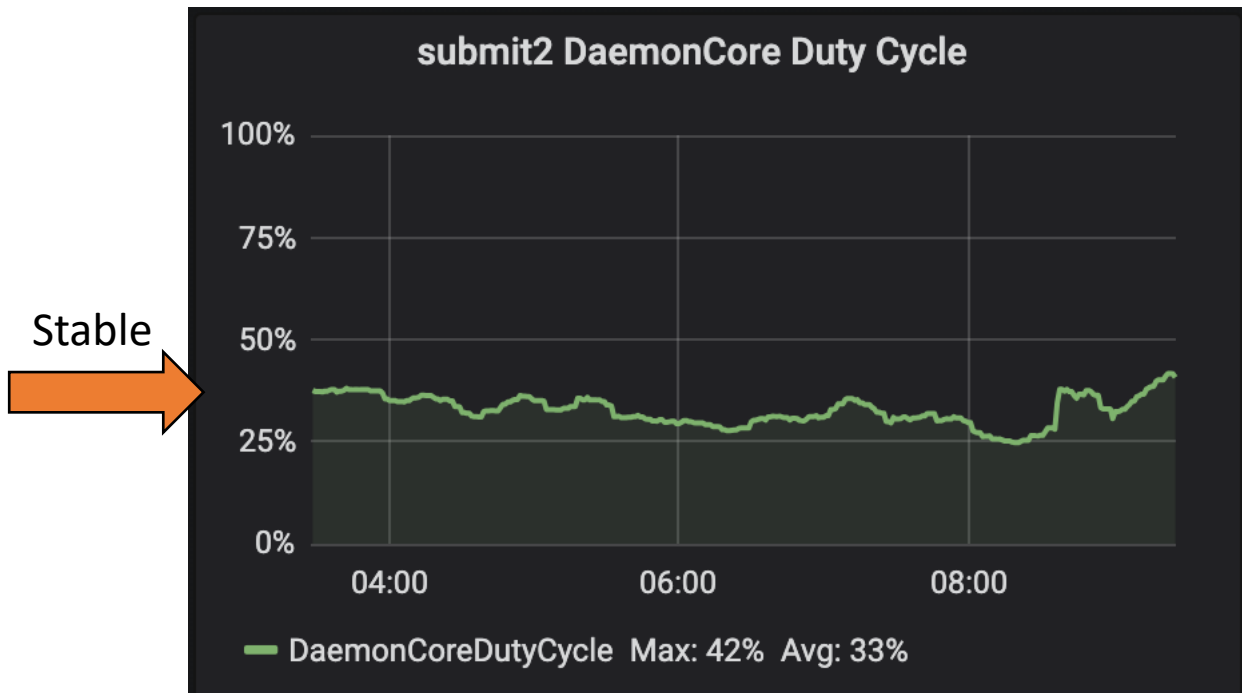
How to be a Responsible HTC Researcher

A meme featuring Woody and Buzz Lightyear from the movie Toy Story. Woody is on the left, looking slightly concerned. Buzz is on the right, wearing his green and purple space suit and holding a purple ring on his right hand. The background is a simple room with a door and a window.

TESTS

TESTS EVERYWHERE

Responsible HTC Researchers



Daemon core duty cycle is a representation of how "busy" the scheduler is with all its responsibilities.

In General

- **Stable user experience!**
 - Can easily log in, jobs can have fewer access point related issues, etc.
 - Allows you and other researchers to carry out your work with little interruption
- **Makes IT staff happy!**
 - Allows them to focus more on things such as deploying new features, system upgrades, etc.

Don't be afraid to think big with your computing!

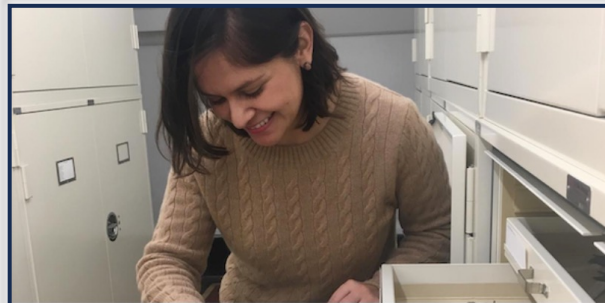
HTCondor systems are designed to efficiently manage many jobs and we are here to *support you* to use computing resources to do novel research.



OSG helps LIGO scientists confirm Einstein's unproven theory

February 11, 2016

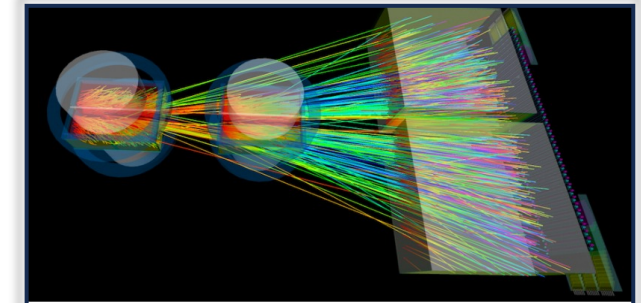
LIGO's detectors search for gravitational waves from deep space. With two detectors, researchers can use differences in the wave's arrival times to constrain the source location in the sky. LIGO's first



80,000 jobs, 40 billion base pairs, and 20 bats — all in 4 weeks

October 26, 2021

An evolutionary biologist at the AMNH used HTC services provided by the OSG to unlock a genomic basis for convergent evolution in bats.



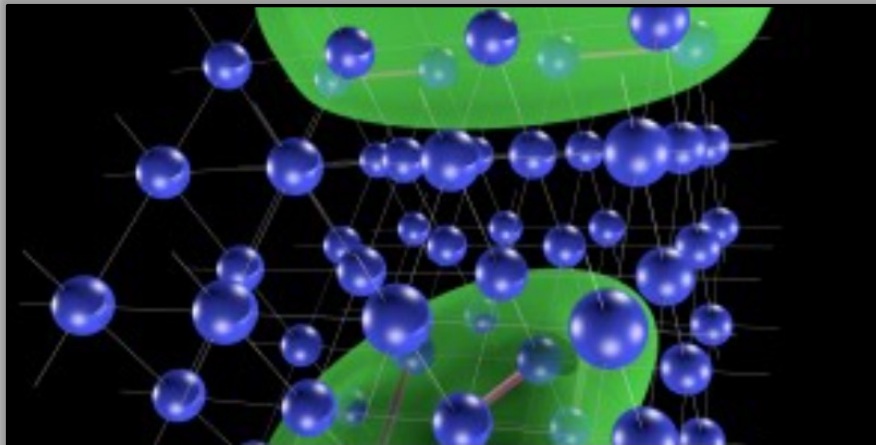
Antimatter: Using HTC to study very rare processes

August 19, 2021

Anirvan Shukla, a User School participant in 2016, spoke at this year's Showcase about how high throughput computing has transformed his research of antimatter in the last five years.

But be conscious of your code, job structure, and of the staff helping you complete your research.

Questions?



Tackling Strongly Correlated Quantum Systems on OSPool

April 10, 2015

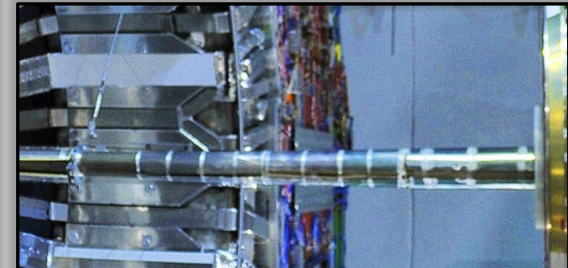
Duke University Associate Professor of Physics Shailesh Chandrasekharan and his graduate student Venkitesh Ayyar are using the OSPool to tackle notoriously difficult problems in quantum systems.

OSPool: Serving Open Science throughput computing

On August 7

552K jobs completed

Placed by 70 researchers
Triggering 6M file transfers
Consuming 1M core hours



Distributed Computing at the African School of Physics 2022 Workshop

April 24, 2023

Over 50 students chose to participate in a distributed computing workshop from the 7th biennial African School of Physics (ASP) 2022 at Nelson Mandela University in Gqeberha, South Africa.

Additional Slides of Interest



Other Thoughts: What's Your Context?

Major considerations (details vary a lot!):

- Computational workflow: Automate running of different jobs in order
- Technical workflow: Management of runs, files, etc.
- Research workflow: Revisit goals, scale, and how computing fits



Other Thoughts: Computational Workflows

- Do you have different kinds of jobs that need to be run in a specific order to implement your overall computational goal?
- If so, you may be able to use HTCondor DAGMan (or other tools) to automate parts or all of the whole process.
- Attend the optional DAGMan lecture later today if you think this situation applies to you!
- **Test computational workflows with few jobs of each type, just as you would in Stage 2.**



Other Thoughts: Technical Workflow Considerations

- How will you get the necessary files to the Access Point and, if needed, OSDF Origin?
- How will you manage individual sets of runs? For example, how will you organize files (see earlier)? If you need to rerun jobs, how will you keep track of each run?
- How will you move results off of the Access Point (which is temporary storage **only**)? Do you have a place to archive results? How will it be organized?



Other Thoughts: Research Workflow Considerations

Now that you have run some jobs:

- Consider the balance between human effort (yours!) and computer time; will the use of HTC actually save you time in the long run and improve your research?
- Estimate how much total calendar time it will take your computational work to complete. Do you have enough time before your next deadline?
- Could you do even more? If things are going well, could you expand your research questions by using more computing? Think big!