

Day 12: Scripting Workflows I

Parameter Sweeps

Turn In Homework

Homework Review

B240 Tour

Scripting Workflows

The Need for Scripting

- Since Condor runs jobs and manages workflows, why do we even need to script anything?
- Jobs are usually part of much larger workflow
 - Instruments → data → *jobs* → results → papers → funds!
 - Human tasks: Design experiments, interpret results, ...
 - Scripting can assist in these steps
- But even for the jobs...
 - Beforehand: Prepare workflow, jobs, data
 - Afterward: Handle data, clean up

Example

- Queue simulator! Say, for the UW Credit Union
- Vary:
 - Number of tellers:1–3
 - Arrival rate: 1–60 per hour
 - Allow departures or not
- 360 combinations
 - Each combination is one set of command-line args.
 - 360 **arguments** and **queue** statements
- Do *you* want to set up and submit jobs manually?

Parameter Sweeps

Parameter Sweeps Defined

- Run same code for a range of input values
- Combinations of multiple ranges (n dimensions)
- Defining ranges
 - *Start - stop - step*
 - Are boundaries included?
 - E.g.: 1 to 1000 2–256, evens [40.0, 80.0) by 0.25
 - *Start - stop - count*
 - Are boundaries included?
 - E.g.: 1000 runs from 40.0 up to 60.0
 - *Start - count - step*
 - E.g.: 1000 trials starting at 1200 counting by 10s

Parameter Sweep in One Dimension

- Enumerating all values of a numeric range is easy
- Convert to *start-stop-step* and use `xrange()`
 - Arguments are integers, can convert to float in loop
 - Range stops before *stop*
 - *start* defaults to 0; *step* defaults to 1, can be < 0

```
for i in xrange(start, stop, step):  
    # Calculate real value, if needed  
    # Do something with value
```

- Non-numeric ranges use sequences: `list`, `file`, ...

```
for i in list_of_values:
```

Parameter Sweep in Many Dimensions

- Use nested loops:

```
for i in parameter_1:  
    for j in parameter_2:  
        for k in parameter_3:  
            # Calculate real value(s)  
            # Do something with values
```

- Bank queue example:

```
for tellers in xrange(1, 4):  
    for rate in xrange(2, 120):  
        real_rate = rate / 2.0  
        run_queue_sim(tellers, real_rate)
```

Data-Driven Code

- Writing loops is easy
- But what happens when you change your design?
- Consider writing generic parameter sweep code
- Actual parameter ranges come from file
- Changing parameters = changing a text file
- This is an example of *data-driven* code:
 - Write general purpose code
 - Vary behavior from outside (files, arguments)
 - Spend less time changing code to use
 - But... make only as general as you need now!

Data-Driven Parameter Sweeps I

- First, design format of parameters in file

```
# Very simple: Bank queue parameters  
tellers, 1, 4, 1  
rates, 1.0, 60.0, 0.5
```

- Getting fancier:
 - Different types of parameters
 - More human-friendly syntax

```
# Way more complex  
site: "Site Code" = {ABC, DEF, GHI, JKL}  
power: "Power Factor" = [0.0, 1.0) x 0.05  
gain: "Gain" = 1-4 ((10 ** _))
```

Data-Driven Parameter Sweeps II

- Write code to read and parse parameter file
- Create sequences for each parameter

```
params = []
for line in param_file:
    parts = re.split(r'\s*,\s*', line)
    start = int(parts[1])
    stop = int(parts[2])
    step = int(parts[3])
    p_range = xrange(start, stop, step)
    params.append((parts[0], p_range))
```

Data-Driven Parameter Sweeps III

- Use iterator function to visit every combination:
`itertools.product()` (Python ≥ 2.6), else:

```
def product(*args):
    pools = map(tuple, args)
    result = [[]]
    for pool in pools:
        result = [x + [y] for x in result for y in pool]
    for prod in result:
        yield tuple(prod)
```

```
# pN is parameter range in sequence type
for t in itertools.product(p1, p2, p3):
    # t is tuple of parameter values
    # Do stuff with this combination
```

Condor

Overview of Approaches

Assuming that we want to run a Condor job for each combination of parameter values, ...

1. Separate submits
2. One submit, many **arguments** & **queue** statements
3. One submit, many directories

Separate Submit Files

- How it works:
 - For each combination of parameter values:
 - Write a Condor submit file with all necessary lines
 - Parameter values: **arguments** statement or input file
- Disadvantages
 - Must submit each job separately
 - Extra overhead
 - Leaves many submit files around

Parameters in Arguments I

- Overview
 - Script creates one huge submit file
 - Each parameter combo gets **arguments & queue** lines
 - Input, output, error, and log files:
 - ✦ All in same directory; files named with **\$(process)**
 - ✦ Each in separate directory per **\$(process)**

```
...  
arguments "1 20"  
queue  
arguments "1 21"  
queue  
...
```

Parameters in Arguments II

- Put all of the common submit statements in a file:

```
# submit-prefix.txt  
executable = sweep.py  
universe = vanilla  
output = sweep-out/sweep-$(PROCESS).out  
error = sweep-err/sweep-$(PROCESS).err  
log = sweep-log/sweep-$(PROCESS).log  
  
should_transfer_files = YES  
when_to_transfer_output = ON_EXIT
```

Parameters in Arguments III

```
# Sketch of main script to make submit file

header = read_submit_prefix()      # string
submit = open(filename, 'w')
submit.write(header)

params = read_parameters_file() # from earlier
for t in product(*params):
    args = ' '.join(t)
    submit.write('arguments = "%s"\n' % args)
    submit.write('queue\n')
submit.close()

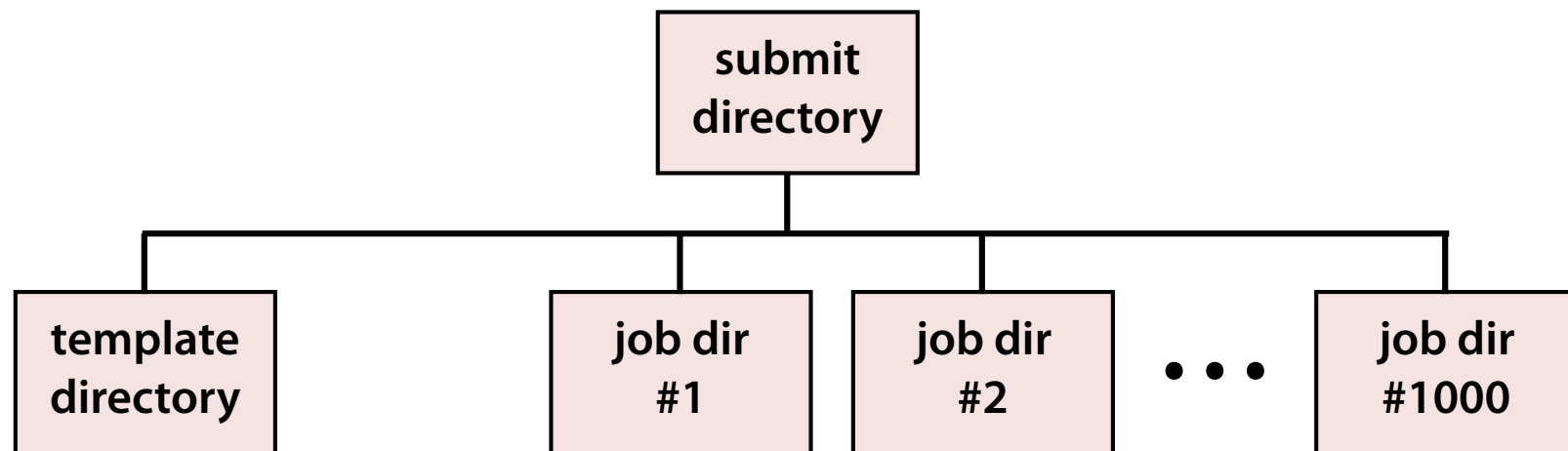
if options.submit:
    print 'Submitting job...'
    os.system('condor_submit ' + filename)
```

Arguments vs. Files

- Parameters via *command-line arguments*
 - When you must, because of the program
 - For few and/or simple parameters
- Parameters via *input files*
 - When you must, because of the program
 - For complex parameters
 - When you must use input files for other reasons

Parameters in Files I

- How it works:
 - Manually write one submit file (details on next slide)
 - For each combination of parameter values, script:
 - ✦ Creates a numbered subdirectory
 - ✦ Writes template files, possibly modified, into directory
 - ✦ Like homework assignment #6



Parameters in Files II

- Write one submit file for all jobs
- Use **initialdir** with **\$(PROCESS)** for job subdirs
- Put **queue N** at end (script should modify **N**)

```
executable = file-sweep.py
universe = vanilla
initialdir = sweep-$(PROCESS)
output = sweep.out
error = sweep.err
log = sweep.log
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = params.txt, ...
queue 1000
```


Parameters in Files III

- Need good function to write modified template file
- Pick parameter placeholder text to avoid conflicts

```
# Outline of a template writer function
# params: (('p1', 42), ('p1', 43), ...)
def write_template(text, target_name, params):
    for p in params:
        p_name, p_value = p
        p_src = '{:%s:}' % p_name
        text = text.replace(p_src, p_value)
    output_file = open(target_name, 'w')
    output_file.write(text)
    output_file.close()
```

Parameters in Files IV

- Read files from template dir into, say, dictionary
- Then, make all directories and files for run

```
# Outline of code to prepare a template run  
# sources: dict from filename to contents  
def write_job_dirs(sources, count, params):  
    for i in xrange(count): # [0, count)  
        dirname = 'sweep-' + str(i)  
        os.mkdir(dirname)  
        pfile = os.path.join(dirname, 'params.txt')  
        write_parameters(params, pfile)  
        for filename in sources:  
            text = sources[filename]  
            target = os.path.join(dirname, filename)  
            write_template(text, target, params)
```

Parameters in Files V

- Top-level plan: Read data, write directories and files
- Could also submit Condor job

```
# Outline of main script  
  
opts, args = parse_command_line()  
  
params = read_parameters(args['param_path'])  
sources = read_sources(args['template_dir'])  
  
update_queue_n(params)  
write_job_dirs(sources, count, params)  
  
if opts.submit:  
    os.system('condor_submit sweep.sub')
```

Output

Harvesting Output

- Need a post-script to gather or consolidate output?
- Without DAGMan, no post-script in job per se
- If this is significant work, use a separate job!

```
# Assumes all interesting output is from stdout  
for outfile in glob.glob('sweep-*/sweep.out'):  
    handle_output(outfile)
```

```
outfiles = ('sweep.out', 'out1.txt', 'out2.csv')  
regexp = r'sweep-\d+$'  
for d in os.listdir('.'):  
    if (os.path.isdir(d) and re.match(regexp, d)):  
        for outfile in outfiles:  
            handle_output(d, outfile)
```

Consolidating Output

- Combine all output files into one
- Prefix each line with parameter info
- Need way to recover parameters for each run
(maybe write to file when creating run directory?)

```
combo = open(combined_output_filename, 'w')  
for d in run_directories:  
    d_params = read_run_params(d)  
    d_output = read_run_output(d)  
    for line in d_output:  
        combo.write('\t'.join(d_params) + '\t')  
        combo.write(line)  
combo.close()
```

Cleaning Up

- If running many times, clean up after each run

```
# Just a sketch of some possibilities
```

```
def handle_output(dir, file):  
    out_path = os.path.join(dir, file)  
    n = re.match(r'sweep-(\d+)$', dir)[1]  
    new_file = 'output-%d.txt' % (n)  
    new_path = os.path.join('output', new_file)  
    shutil.move(out_path, new_path)  
  
for run_dir in run_directories:  
    handle_output(run_dir, 'sweep.out')  
    shutil.rmtree(run_dir)
```

Homework

Homework

- Make a pretty picture!
- Instead of one long job, break into several smaller jobs... by “tile”
- Lots of background info, read carefully to understand
- For now, stitch resulting image files together manually

