Getting the Most out of HTC with Workflows
Friday morning, 9:00 am

Christina Koch ckoch5@wisc.edu
Research Computing Facilitator
University of Wisconsin - Madison
Why are we here?
Why are we here?

To do SCIENCE!!!

- A lot of science is best-done with computing – sometimes, LOTS of computing
- Science needs to be reproducible
- And, we’d really like science to happen fast(er)
GETTING THE MOST OUT OF COMPUTING (FOR RESEARCH)
At the beginning of the week, we talked about two different approaches for tackling large compute tasks...

- high-throughput
- high-performance (e.g. MPI)
Two Strategies

High Throughput
Focus: Workflows with many small, largely independent compute tasks
Optimize: *throughput*, or time from *submission* to overall completion

High Performance
• Focus: Workflows with large, highly coupled tasks
• Optimize: *individual tasks*, software, communication between processes
Making Good Choices

- How do you choose the best approach?
- Guiding question:

  Is your problem “HTC-able”?
Typical HTC Problems

- batches of similar program runs (>10)
- “loops” over independent tasks
- others you might not think of …
  - programs/functions that
    - process files that are already separate
    - process columns or rows, separately
    - iterate over a parameter space
  - a lot of programs/functions that use multiple CPUs on the same server

Ultimately: Can you break it up?
What is not HTC?

- fewer numbers of jobs
- jobs individually requiring significant resources
  - RAM, Data/Disk, # CPUs, time
  (though, “significant” depends on the HTC compute system you use)
- restrictive licensing
The Real World

• However, it’s not just about finding the right computing approach to your problem.

• These approaches will be *most* effective if they’re running on appropriate compute systems.
The Real World

• Not all compute systems are created equal.

• Two questions to ask:
  What resources are available to me?
  Which one is the best match for the kind of computing I want to do?
Campus Resources

• Start with your local campus compute system
• Some considerations:
  – Who has access? Are there allocations?
  – What kind of system? What is it optimized for?
• An HPC cluster may not handle lots of jobs well, in the same way that an HTC system has limited multicore capabilities - be aware of how a system matches/doesn’t match your computation strategy.
• Ask questions! Be a good citizen!
• If local resources are limited, explore other options.
Beyond your campus

• Open Science Grid!
  - This afternoon, Tim will talk about ways to access OSG after the school is over

• Other grids
  - European Grid Infrastructure
  - Other national and regional grids
  - Commercial cloud systems
The payoff

- HTC is, beyond everything, scalable
  - If you can run 10 jobs, you can run 10,000, maybe even 10 million
- Worth pursuing the right kind of resources (if you can) for the right kind of problem.
GETTING THE MOST OUT OF HTC
Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible
Throughput, revisited

• In HTC, we optimize *throughput*: time from submission to overall completion
• Instead of making individual jobs as fast as possible, optimize how long it takes for all jobs to finish.
• We do this by breaking large processes into smaller pieces
Breaking up is hard to do…

• Ideally into parallel (separate) jobs
  - reduced job requirements = more matches
  - not always easy or possible

• Strategies
  - break HTC-able steps out of a single program
  - break up loops
  - break up input

• Use self-checkpointing if jobs are too long
Batching (Merging) is easy

• A single job can
  – execute multiple independent tasks
  – execute multiple short, sequential steps
  – avoid transfer of intermediate files

• Use scripts!
  – need adequate error reporting for each “step”
  – easily handle multiple commands and arguments
1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible
Know and Optimize Job Use of Resources!

- **CPUs** (“1” is best for matching; essential for OSG)
  - restrict, if necessary/possible
  - software that uses all available CPUs is BAD!

- **CPU Time**
  - > ~5 min, < ~1 day; **Ideal: 1-2 hours**

- **RAM** (not always easily modified)

- **Disk** per-job (execute) and in-total (submit)

- **Network Bandwidth**
  - minimize transfer: filter/trim/delete, compress
Use the job log

001 (2576205.000.000) 06/07 11:57:57 Job executing on host: <128.104.101.248:9618>
005 (2576205.000.000) 06/07 14:12:55 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

5 - Run Bytes Sent By Job
104857640 - Run Bytes Received By Job
5 - Total Bytes Sent By Job
104857640 - Total Bytes Received By Job

Partitionable Resources:

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Usage</th>
<th>Request</th>
<th>Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpus</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Disk (KB)</td>
<td>122358</td>
<td>125000</td>
<td>13869733</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>30</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible
Bring *What* with You?

- Software (covered Wednesday)
- Data and other input files
  - Parameters and random numbers: generate and record ahead of time (for reproducibility)

- What else?
Wrapper Scripts are Essential

• Before task execution
  - transfer/prepare files and directories
  - setup/configure software environment and other dependencies

• Task execution
  - prepare complex commands and arguments
  - batch together many ‘small’ tasks

• After task execution
  - filter/combine/compress files and directories
  - check for and report on errors
Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible
Testing, testing, testing!

- Will be a major focus of our exercises today.
- Allows you to optimize resource use (see HTC tactic #2)
- Just because it worked for 10 jobs, doesn’t mean it will work for 10,000 jobs (scaling issues)
  - Data transfer (in and out)
  - Discover site-specific problems
Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
3. Bring Dependencies With You
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible
What to Automate?

- Submitting many jobs (using HTCondor)
- Writing submit files using scripts
- Running a series of jobs, or workflow
What is a workflow?

• A series of ordered steps
  • Steps
  • Connections
  • (Metadata)
We ❤ workflows

• non-computing “workflows” are all around you, especially in science
  – instrument setup
  – experimental procedures and protocols

• when planned/documentated, workflows help with:
  – organizing and managing processes
  – saving time with automation
  – objectivity, reliability, and reproducibility
    (THE TENETS OF GOOD SCIENCE!)
DAGs Automate Workflows

- Data prep/split
- Process '0'
- Process '99'
- Combine, assess results

---

OSG User School 2016
Automating workflows can save you time...

Here is a chart that illustrates how long you can work on making a routine task more efficient before you're spending more time than you save, across five years.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Second</td>
<td>1 Day</td>
<td>2 Hours</td>
<td>30 Minutes</td>
<td>4 Minutes</td>
<td>1 Minute</td>
<td>5 Seconds</td>
</tr>
<tr>
<td>5 Seconds</td>
<td>5 Days</td>
<td>12 Hours</td>
<td>2 Hours</td>
<td>21 Minutes</td>
<td>5 Minutes</td>
<td>25 Seconds</td>
</tr>
<tr>
<td>30 Seconds</td>
<td>4 Weeks</td>
<td>3 Days</td>
<td>12 Hours</td>
<td>2 Hours</td>
<td>30 Minutes</td>
<td>2 Minutes</td>
</tr>
<tr>
<td>1 Minute</td>
<td>8 Weeks</td>
<td>6 Days</td>
<td>1 Day</td>
<td>4 Hours</td>
<td>1 Hour</td>
<td>5 Minutes</td>
</tr>
<tr>
<td>5 Minutes</td>
<td>9 Months</td>
<td>4 Weeks</td>
<td>6 Days</td>
<td>21 Hours</td>
<td>5 Hours</td>
<td>25 Minutes</td>
</tr>
<tr>
<td>30 Minutes</td>
<td>6 Months</td>
<td>5 Weeks</td>
<td>5 Days</td>
<td>1 Day</td>
<td>2 Hours</td>
<td>1 Day</td>
</tr>
<tr>
<td>1 Hour</td>
<td>10 Months</td>
<td>2 Months</td>
<td>10 Days</td>
<td>2 Days</td>
<td>5 Hours</td>
<td>1 Day</td>
</tr>
<tr>
<td>6 Hours</td>
<td>2 Months</td>
<td>2 Weeks</td>
<td>2 Days</td>
<td>1 Day</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 Day</td>
<td>8 Weeks</td>
<td>5 Days</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... but there are even more benefits of automating workflows

- Reproducibility
- Building knowledge and experience
- New ability to imagine greater scale, functionality, possibilities, and better SCIENCE!!
GETTING THE MOST OUT OF WORKFLOWS, PART 1
From schematics…
... to the real world
Building a Good Workflow

1. Draw out the *general* workflow
2. Define details (test ‘pieces’ with HTCondor jobs)
   - divide or consolidate ‘pieces’
   - determine resource requirements
   - identify steps to be automated or checked
3. Build it modularly; test and optimize
4. Scale-up gradually
5. Make it work consistently
6. What more can you automate or error-check?

(And remember to document!)
Workflow, version 1

1. data prep (minutes)
2. processing (days)
3. assess results (minutes)
Workflow, version 2 (HTC)

- data prep/split
- process ‘0’
- process ’99’
- combine, assess results
Building a Good Workflow

1. Draw out the *general* workflow

2. Define details (test ‘pieces’ with HTCondor jobs)
   - divide or consolidate ‘pieces’
   - determine resource requirements
   - identify steps to be automated or checked

3. Build it modularly; test and optimize

4. Scale-up gradually

5. Make it work consistently

6. What more can you automate or error-check?

(And remember to document!)
Determine Resource Usage

• Run locally first
• Then get one job running remotely
  ▪ (on execute machine, not submit machine)!
  ▪ get the logistics correct! (HTCondor submission, file and software setup, etc.)
• Once working, run a couple of times
  – If big variance in resource needs, should you take the…
    Average? Median? Worst case?
End Up with This

- **process '0'** (filter output)
  - 100 MB RAM
  - 500 MB Disk
  - 40 min (each)
  - (POST-RETRY)

- **process '99'** (filter output)
  - 300 MB RAM
  - 1 GB Disk
  - 15 min
  - (POST-RETRY)

- combine, transform results
  - 300 MB RAM
  - 1 GB Disk
  - 15 min
  - (POST-RETRY)

- **(special transfer)**
  - file prep and split
  - 1 GB RAM
  - 2 GB Disk
  - 1.5 hours
  - (POST-RETRY)
Exercise 1.1

Task 1
(days)

Task 2
(minutes)
Exercise 1.1

process ‘0’ → Task 2 → process ‘?’
Exercise 1.2

? RAM
? MB Disk
? min
(each)

process ‘0’
(filter output)

(process output)

process ‘?’
(filter output)

? RAM
? GB Disk
? min

Task 2

(POST?)

(POST?)

(POST?)
Questions?

• Now: “Joe’s Workflow” Exercise 1.1, 1.2
  – In groups of 2-3
  – Read carefully!

• Later:
  – Lecture: From Workflow to Production
  – Exercises 2.1, 2.2