Optimizing Your Computing

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)
Computing Types

• 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?

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 software 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?
TWO EXAMPLES: LOCAL HTC AND OSG
# CHTC Recommendations

<table>
<thead>
<tr>
<th></th>
<th>Ideal Jobs</th>
<th>Still very advantageous</th>
<th>Less-so, but maybe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpus (Gpus)</td>
<td>1</td>
<td>&lt;20</td>
<td>&gt;20 cpus, using multiple nodes</td>
</tr>
<tr>
<td>Walltime</td>
<td>&lt;12 hours* or checkpointable</td>
<td>&lt;24 hours* or checkpointable</td>
<td>up to 2 weeks</td>
</tr>
<tr>
<td>RAM</td>
<td>1-2GB</td>
<td>up to TBs</td>
<td>&gt;4TB</td>
</tr>
<tr>
<td>Input</td>
<td>&lt;100MB</td>
<td>up to TBs</td>
<td>N/A</td>
</tr>
<tr>
<td>Output</td>
<td>&lt;4GB</td>
<td>up to TBs</td>
<td>N/A</td>
</tr>
<tr>
<td>Software</td>
<td>‘portable’</td>
<td>anything else that’s not licensed, non-Linux</td>
<td></td>
</tr>
</tbody>
</table>
# OSG Recommendations

<table>
<thead>
<tr>
<th></th>
<th>Ideal Jobs! (up to 10,000 cores across jobs, per user!)</th>
<th>Still Very Advantageous!</th>
<th>Less-so, but maybe</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cores (GPUs)</strong></td>
<td>1 (1; non-specific type)</td>
<td>&lt;8 (1; specific GPU type)</td>
<td>&gt;8 (or MPI) (multiple)</td>
</tr>
<tr>
<td><strong>Walltime</strong></td>
<td>&lt;12 hrs*</td>
<td>&lt;24 hrs*</td>
<td>&gt;24 hrs</td>
</tr>
<tr>
<td></td>
<td>*or checkpointable</td>
<td>*or checkpointable</td>
<td></td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>&lt;few GB</td>
<td>&lt;10s GB</td>
<td>&gt;10s GB</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>&lt;500 MB</td>
<td>&lt;10 GB</td>
<td>&gt;10 GB</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>&lt;1 GB</td>
<td>&lt;10 GB</td>
<td>&gt;10 GB</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td>‘portable’ (pre-compiled binaries, transferable, containerizable, etc.)</td>
<td>most other than</td>
<td>Licensed software; non-Linux</td>
</tr>
</tbody>
</table>

**OSG User School 2019**
WHAT ABOUT YOUR LOCAL COMPUTE CENTER?
Campus Resources

• Check out 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
The HTC Goal:

RUN AS MANY JOBS AS POSSIBLE ON AS MANY COMPUTERS AS POSSIBLE
Key HTC Tactics

1. Increase Overall Throughput
   Optimize for total work, not individual jobs
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 (to have more simultaneous processing power)
Breaking Up

• Break work 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

• Consider grouping tasks if jobs are short!
Self-Checkpointing

Solution for long jobs and “shish-kebabs”

1. Changes to your code
   - Periodically save information about progress to a new file (every hour?)
   - At the beginning of script:
     - If progress file exists, read it and start from where the program (or script) left off
     - Otherwise, start from the beginning

2. Change to submit file:

```bash
when_to_transfer_output = ON_EXIT_OR_EVICT
```
Solutions for Larger Files

• File manipulations
  - split input files to **send minimal data** with each job
  - **filter** input and output files to transfer only essential data
  - use compression/decompression

• Follow file delivery methods from yesterday for files that are still “large”
Key HTC Tactics

1. Increase Overall Throughput
2. Utilize Resources Efficiently!
   Jobs will match to more resources
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: 1-10 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 00:00:00:00, Sys 0 00:00:00:00 - Run Remote Usage
Usr 0 00:00:00:00, Sys 0 00:00:00:00 - Run Local Usage
Usr 0 00:00:00:00, Sys 0 00:00:00:00 - Total Remote Usage
Usr 0 00:00:00:00, Sys 0 00:00:00:00 - 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</th>
<th>Usage</th>
<th>Request</th>
<th>Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpus</td>
<td>1</td>
<td>1</td>
<td></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
   Jobs can run anywhere*
4. Scale Gradually, Testing Generously
5. Automate As Many Steps As Possible

*almost
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 Recap

- **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
   Saves you time in the long run!
5. Automate Multiple Steps
Testing, Testing, Testing!

• Will be a major focus of our exercises today.
• Allows you to optimize resource use (see HTC tactic #2), job length (tactic #1)
• Just because it worked for 10 jobs, doesn’t mean it will work perfectly for 10,000 jobs (scaling issues)
  – Data transfer (in and out)
  – Discover site-specific problems
Why test? Imagine if:

• You are using a new scientific instrument. Would you run 100 samples without ever running a test (or a few tests)?

• Your job accidentally creates a 3GB core dump file because the code is corrupted. What happens if you submit 1,000 jobs with this issue?
Testing, Testing, One...

- Get one job working
  - Work out software issues, data transfer patterns, etc.
  - Make subsequent memory/disk requests based on results from this job
  - How long does the job run?
Run a medium/small scale test(s) (10-100 jobs)

- Check memory/disk requests when complete. Are they accurate? If not, change them.
- Do some percentage of jobs fail? If so, can you figure out why? How will you find/handle failures at a larger scale?
- Would it make sense to submit more, smaller jobs or fewer, longer jobs?
- How much data is being generated? Do you have space on the submit server to store the results of the full-scale run?
...Three!

- If you make significant changes in any of the previous testing steps (or make any other changes to your workflow - new code, new data, new version of your software) -- do another quick test.
- Once you’ve done a small and medium test, scale up to the full submission.
Introducing the exercises

• Our exercise today will involve developing a workflow (series of sequential pieces)
• What you need to know for the first two exercises:
  – Identify the component pieces (job submissions) of the workflow and the shape of the overall workflow.
  – Test/optimize the pieces, as described in the previous few slides
Questions?

• Now: “Joe’s Workflow”
  - Exercise 1.1 -- Understand and plan (no jobs)
  - Exercise 1.2 -- Testing jobs
  - Work in groups of 2-3
  - Read carefully!

• Later:
  - Lecture: Optimizing Workflows
  - Exercises 2.1, 2.2