What’s Different About Overlay Systems?

Brian Lin
OSG Software Team
University of Wisconsin - Madison
Overlay Systems are Awesome!

Photo by Zachary Nelson on Unsplash
What’s the Catch?

Requires more infrastructure, software, set-up, management, troubleshooting...
“You know you have a **distributed system** when the crash of a computer you’ve never heard of stops you from getting any work done.”

- Leslie Lamport
#1: Heterogenous Resources

Accounting for differences between the OSG and your local cluster
Sites of the OSG

Source: http://display.opensciencegrid.org/
Heterogeneous Resources - Software

- Different operating systems (Red Hat, CentOS, Scientific Linux; versions 6 and 7)
- Varying software versions (e.g., at least Python 2.6)
- Varying software availability (e.g., no BLAST*)

**Solution:** Make your jobs more portable: OASIS, containers, etc (more in Wednesday’s talks)
Hetero. Resources - Hardware

- CPU: Mostly single core
- RAM: Mostly < 8GB
- GPU: Limited #s but more being added
- Disk: No shared file system (more in Thursday’s talks)

**Solution:** Split up your workflow to make your jobs more high throughput
#2: With Great Power Comes Great Responsibility

How to be a good netizen
Resources You Don’t Own

• Primary resource owners can kick you off for any reason
• No local system administrator relationships
• No sensitive data (again)!

Photo by Nathan Dumlao on Unsplash
Be a Good Netizen!

- Use of shared resources is a privilege
- Only use the resources that you request
- Be nice to your submit nodes

**Solution:** Test jobs on local resources with

```
condor_submit -i
```
#3: Slower Ramp Up

Leasing resources takes time!
Slower Ramp Up

- Adding slots: pilot process in the OSG vs slots already in your local pool
- Not a lot of time (~minutes) compared to most job runtimes (~hours)
  - Small trade-off for increased availability
  - Tip: If your jobs only run for < 10min each, consider combining them so each job runs for at least 30min
Robustify Your Jobs
Succeeding in the face of failure
Job Robustification

- Test small, test often
- Specify output, error, and log files at least while you develop your workflow
- Use on_exit_hold to catch different failure modes
  - on_exit_hold = (ExitCode =?= 3)
  - on_exit_hold = (time() - JobCurrentStartDate < 1 * $(HOUR))
- For jobs that run too long:
  - periodic_hold = (time() - JobCurrentStartDate > 4 * $(HOUR))
  - periodic_release = (HoldReasonCode == 3) && (NumJobStarts < 3)

HoldReasonCode is 3 for any jobs where on_exit_hold or periodic_hold evaluate to True
Job Robustification

• In your own code:
  – Self checkpointing
  – Different exit codes for use with on_exit_hold
  – Defensive troubleshooting (hostname, ls -l, pwd, condor_version in your wrapper script)
  – Add simple logging (e.g. print, echo, etc)
Questions?