Galaxy as an Extensible Job Execution Platform

John Chilton and the Galaxy Team

Slides @ http://bit.ly/bosc2014

Goal...

a developer sales pitch

... to convince you that Galaxy's runs jobs on clusters (& "clouds") awesomely.

... explain whole process is pluggable, existing plugins provide useful functionality, and writing Galaxy components is a great way to potentially receive wide usage and exposure for innovative infrastructure work.

... convince you to leverage Galaxy with your next application.

Leveraging Galaxy

- Write tools for your applications.
  - Examples: Hundreds of application in the Tool Shed
- Extend Galaxy.
  - Examples: Globus Genomics, Medbook, 50+public servers
- Leverage Galaxy via the API.
  - Example: Refinery
- Just run jobs like Galaxy using Pulsar.
  - Example: Your next platform?

Traditional Galaxy Execution Model (Background)

"Tools" describe UI and how to build command lines...

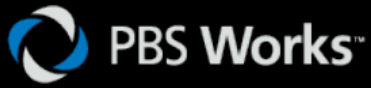
... commands executed on a cluster ...

... deployers describe resource requests for each tool...

... files accessed directly by Galaxy and compute nodes.

We are breaking down these limitations, extending the functionality it exciting new directions, and allowing deployers to leverage cutting edge computational infrastructure.

Where can Galaxy run jobs?



Deployer-Developer Separation

Galaxy allows deployers to describe how to tools get routed to clusters based on job parameters, user, environment, etc....

Makes it easy to submit to many different clusters, heterogeneous resources.

Build your own...

Dynamic plugins (Python class) for how to run jobs on remote resources (runners).

Dynamic plugin infrastructure for how to route job based on whatever (functions).

Dynamic Job Destination Enhancements

- Allow deployers to delay job queuing (enforce custom limits, etc...).
- Deployment specific parameter collection.
- Provide destination plugins with a high-level utility to reason about user's recent resource usage - how many jobs, their runtime, allocations, etc...
  - Easily build your own fair-share prioritization scheme on top of your cluster (or clusters).

Dynamic Job State Handlers

Allow deployers to write simple Python functions to describe how to respond to various job state events.

Key Application: Resubmit jobs to larger resources after walltime or memory limits are hit.

Fun(?) Fact: Nearly half the jobs corresponding a popular NGS mapping tool fail in the first 2 minutes of execution on usegalaxy.org.

Dependencies

Describe by abstract packages or specific containers (i.e. Docker).

- Pluggable, configurable *package* resolution:
  - Traditional Galaxy scripts (e.g. env.sh files)
  - Tool Shed
  - Linux Modules
  - Build your own...
- Pluggable (?) container resolution:

Containers - Docker

http://www.docker.com/whatisdocker/

“lightweight runtime and packaging tool [...] run the same app, unchanged, on laptops, data center VMs, and any cloud”

“enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient”

Docker in Galaxy

- Tools may declare image ids they are compatible with.
- Deployers may specify various destinations have docker enabled and how to utilize it there.
- Deployers may optionally specify container or override tool-supplied container for isolation/security reasons.

Job Metrics

Instrument jobs to collect runtime metrics.
Some simple examples:

- Collect information about job runtime
- Cores allocated
- Compute resource information (CPU, Memory, OS, etc...)
- Environment data

Collectl

http://collectl.sourceforge.net/

Typically, Collectl is run as a daemon and collects ***detailed*** system wide metrics (CPU use, memory use, I/O, etc...).

It is actually highly customizable, efficient, and capable of generating and replaying detailed statistics about per-process resource consumption (contrasting it with more modern, popular alternatives such as collectd).

Job Metrics - Collectl

Many options for how to instrument Galaxy jobs with collectl - but at a high-level can either grab detailed logs about for offline playback collection or it can parse them in after job is complete, filter out processes corresponding to a particular job, and produce a variety of statistics for many different resource utilizations.

Job Metrics

Write your own...

The Shared File System Problem

A traditional limitation of Galaxy was need for a large shared file system with compute nodes.

Not always cost effective (especially for large file stores) or politically feasible, and prevents bursting out to clouds, national computing centers, etc....

Introducing Pulsar

(formerly the LWR)

Briefly mentioned LWR at last BOSC - but it has grown up...

... then it was useful for running jobs on one-off remote servers, today it is running jobs on some of the most well known clusters in the US.

Huge thanks to Nate Coraor!

Pulsar Developments

- New Pluggable backends - mirroring that of Galaxy.
  - DRMAA, Condor, remote CLI submission. Running jobs as “real user”, etc...
- New RESTful API
- New Galaxy features have been implemented with Pulsar support from get go
  - Pluggable dependency resolution, containerization
  - Pluggable job metric instrumentation
  - Allow tools to uniformly access cores count allocation (GALAXY_SLOTS).
  - Environment mod injection

Many benefits of Galaxy without needing a Galaxy database.

Pulsar Message Queues

AMQP
Advanced Message Queuing Protocol

- Opening firewall holes for Pulsar would be problematic at many institutions.
- Allowing Pulsar to be driven via message queue has expanded its utility.
- Built on Kombu - Python interface to the generic AMQP protocol.

Pulsar File Actions

Reality of file systems at large institutions can be complex. Large, slow, backed up disk mounted here, quick scratch disk mounted there, same partitions with different mount points, etc....

Could certainly just transfer everything needed for every job - but Pulsar client also allows staging, rewriting on per path basis and a plugin strategy for defining different staging and data transfer techniques. Optimize data access for your deployment.

Pulsar on Mesos

(Prototype)

Apache MESOS™

- Pulsar can act as a Mesos “framework”.
  - Mesos used by Twitter, Netflix, Vimeo, etc...
  - Less of a resource scheduler than the glue to build a resource scheduler or parallelization framework.
  - Pulsar runs on each compute node - can distribute (Galaxy) jobs across a Mesos cluster without any shared disk at all.

Thanks!

The Galaxy Team

The Galaxy-P grant, team, and the Minnesota Supercomputing Institute for funding initial development of the LWR - with special thanks to Tim Griffin, Pratik Jagtap, Benjamin Lynch, and Anne-Françoise Lamblin.

The Galaxy Community for building awesome stuff with Galaxy and pushing the platform forward.

Questions?

If not, here are a bunch of URLs...

Galaxy

- https://bitbucket.org/galaxy/galaxy-central
- https://github.com/galaxyproject/pulsar
- http://galaxyproject.org/
- http://pulsar.readthedocs.org/
- https://usegalaxy.org/

Other Stuff

- http://www.docker.com/
- http://mesos.apache.org/
- http://www.tacc.org/
- https://iplantcollaborative.org/
- http://www.psc.edu/
- http://www.drmaa.org/

...or just Google it I guess.

Galaxy @ the ISMB

Many talks and posters presentations - find out more at @ bit.ly/gxyismb2014.

Charts, workflows, actual demos - I assure you the rest are more exciting than this was - so check ‘em out!

Abandoned Slides....

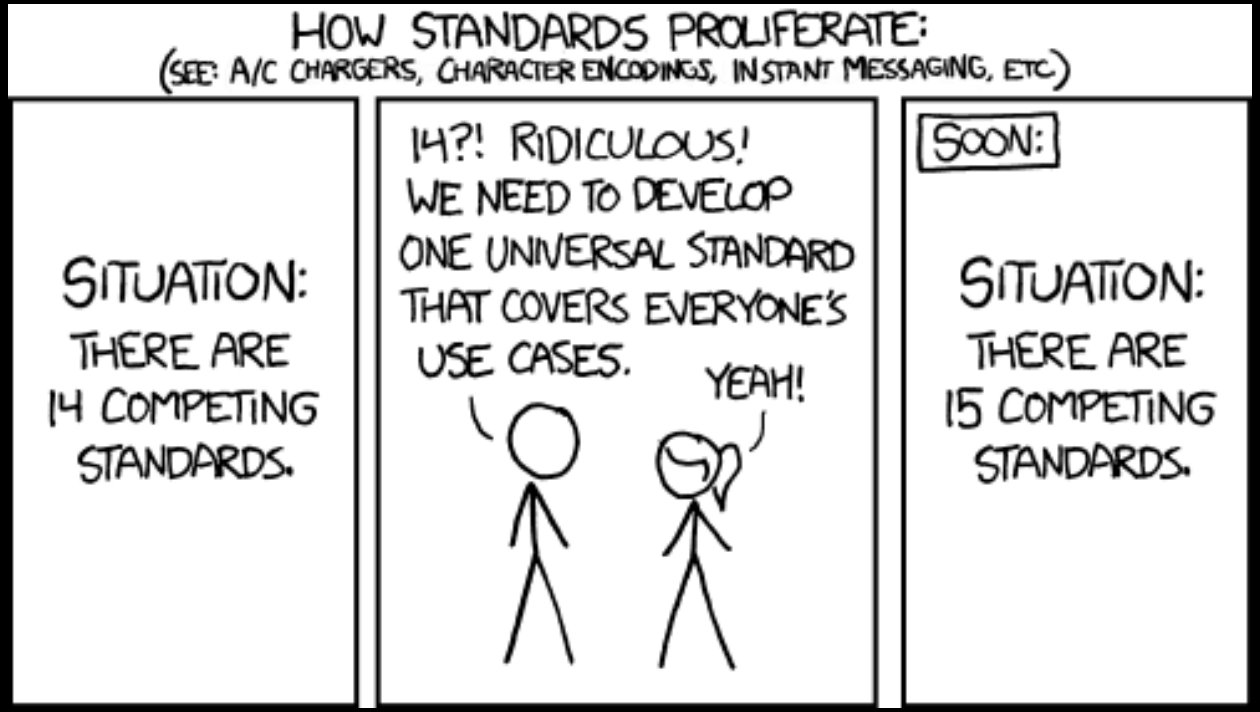
~~Galaxy as an Extensible Job Execution Platform~~ Aspect-Oriented Clusters with Galaxy

John Chilton and the Galaxy Team

What my goal is not...

... that you can leave here and configure a Galaxy instance, write a Galaxy tool, or use the API.

Mandatory XKCD...



http://xkcd.com/927/