UF Galaxy Implementation

Oleksandr Moskalenko, Ph.D.
University of Florida Research Computing

Super Computing 2012, Salt Lake City, Nov 12-15, 2012

Introduction

- About Galaxy
- Structure of the Galaxy instance
- Storage
- Computing hardware
- Software
- References
- User handling and user job submission

Galaxy goals

- Provide an easy to use interface for entry level analyses at first, expand the mission as the Galaxy's capabilities grow
- Provide a collaborative environment
- Provide a teaching environment
- Supplant the iNquiry instance to be retired from the Genetics Institute

History

- 1st prototype – KVM Virtual Machines
- 2nd prototype – head node, Lustre, dedicated Torque/MOAB reservation
- 1st production setup – (10 months)
  - Head node (4cores, 16GB RAM)
  - Cluster (SGE – 4 12-core/96GB RAM nodes)
  - Lustre FS (60TB)
  - 1Gbit network, Infiniband from nodes to storage
  - Software provided by the modules system

Current production setup

- > 220 users, > 12000 jobs @ about 1-2k jobs per month
- Hardware:
  - Head node – 4 cores, 16GB RAM → 32GB upcoming
  - Cluster
    - 7000 cores, 2GB/core average RAM
    - 120, 500GB RAM bigmem nodes (1TB upcoming)
    - Torque/MOAB PBS
  - Storage
    - Current – NexentaStor ZFS/NFS – slow
    - Upcoming – HA NexentaStor ZFS/NFS – fast

Production Configuration

- Authentication – Remote User
  - Apache mod_auth_tkt, LDAP back-end.
- Software – modules and Galaxy main tool shed
- Database node – 4 core, 32GB RAM, PostgreSQL
- Local (campus) Tool Shed

Current production setup

- 3 head nodes: production, staging, development
  - Development: Up-to-date galaxy-central code, Wrapper development, galaxy core work
  - Staging:
    - Pull galaxy-dist releases
    - Pull and update a copy of the production database
    - Clean and stabilize
    - Set up tools and local modifications and wrappers
    - Replace the production code

Initial Hurdles

- Torque - python_pbs memory leaks
- Head node overloaded
- Poor visualization capabilities
- No soft restart capability
- Upgrades – tool cleanup, maintaining hacks
- Providing consistent environment (drmaa.py / modules)
- No job accounting
- “I can haz moar wrapperzzz”

Early Hacks

- Watcher cron job for python_pbs
- Run **all** jobs on the cluster, even the upload
- Soft restart init scripts
- Use two RCS – mercurial and git to manage upgrades
- Hacked drmaa.py to load modules
- Unified the tools used on the cluster and in Galaxy via the environment modules system
- Unified reference databases
- Set up big data import

User Issue Report Categories

- Inappropriate job resource requests hard-coded into the tool runner URIs
- Tool failures because of bad data or tool bugs
- Poor understanding of what tools are needed
- Help with analyses
- Reference dataset issues (dbkey)

Move to real user jobs

- Jobs must run under the real user's id
  - Very powerful feature, but opens many cans of worms
- Move from SGE to Torque/MOAB
  - No green banner of death
  - Strict resource request handling
  - Finding a drmaa library that works –
- Introduction of the Tool Shed
  - Cultivate new wrapper contributors
  - Crucial divergence from the modules in version handling
- Dynamic job runner
  - Almost unlimited possibilities in with the real userid jobs

The Wish List

- Universal job resource request interface mechanism
- More flexible output file handling
- No more hard-coding tool arguments
- Easier access to core developers
- Reference dataset handling automation
- More documentation and more capabilities in the tool definition file logic
- More published workflows and docs