ATLAS CONNECT

VISION

Leverage robust, off-the-shelf components from the OSG Campus Grids program, HTCondor, Globus, CI Connect, AutoPyFactory, CCTools and the BNL RACF to provide a hybrid job submission and data management service which integrates directly with existing ATLAS systems such as PanDA, FAX and Rucio. The new services will focus on providing connective services to optimize ease of use by end-user physicists, and to maximize resource capacity by bridging to shared resources on university campuses currently outside the program-funded US ATLAS Distributed Computing Facility. These services include a federated identity management system for secure authentication using credentials provided by a user's home institution, and a group management system for authorized access to protected resources. Job submission services will provide capability for direct submission via HTCondor to natively scheduled or flockable HT Condor clusters, submission using PanDA pilot factory systems and central database systems, and submission using Campus HTCondor glide-in services such as BOSCO. Resources principally targeted are beyond pledge Tier 1 and Tier 2 resources, Tier 3 clusters, and shared campus facilities such as a university HPC center. The service should benefit the entire US ATLAS physics community providing a common inter-campus resource sharing platform, a job submission and data service environment easily configured to reach a variety of resource types from a single location. For example, a physicst with an allocation on his university's condo cluster, perhaps obtained through startup funding from his Dean, can schedule jobs to this resource alongside submission to the production grid, a dedicated, institutional Tier 3 center, any campus HPC center participating in ATLAS Connect service, or to another institutions private resources, all using the same /home directory. Finally, ATLAS Connect will inherit developments in provisioning services to agile infrastructures advances as part of the US ATLAS Facilities program, including work from BNL RACF AutoPyFactory, OSG Technology Investigations, CERN IT, Globus and elsewhere.

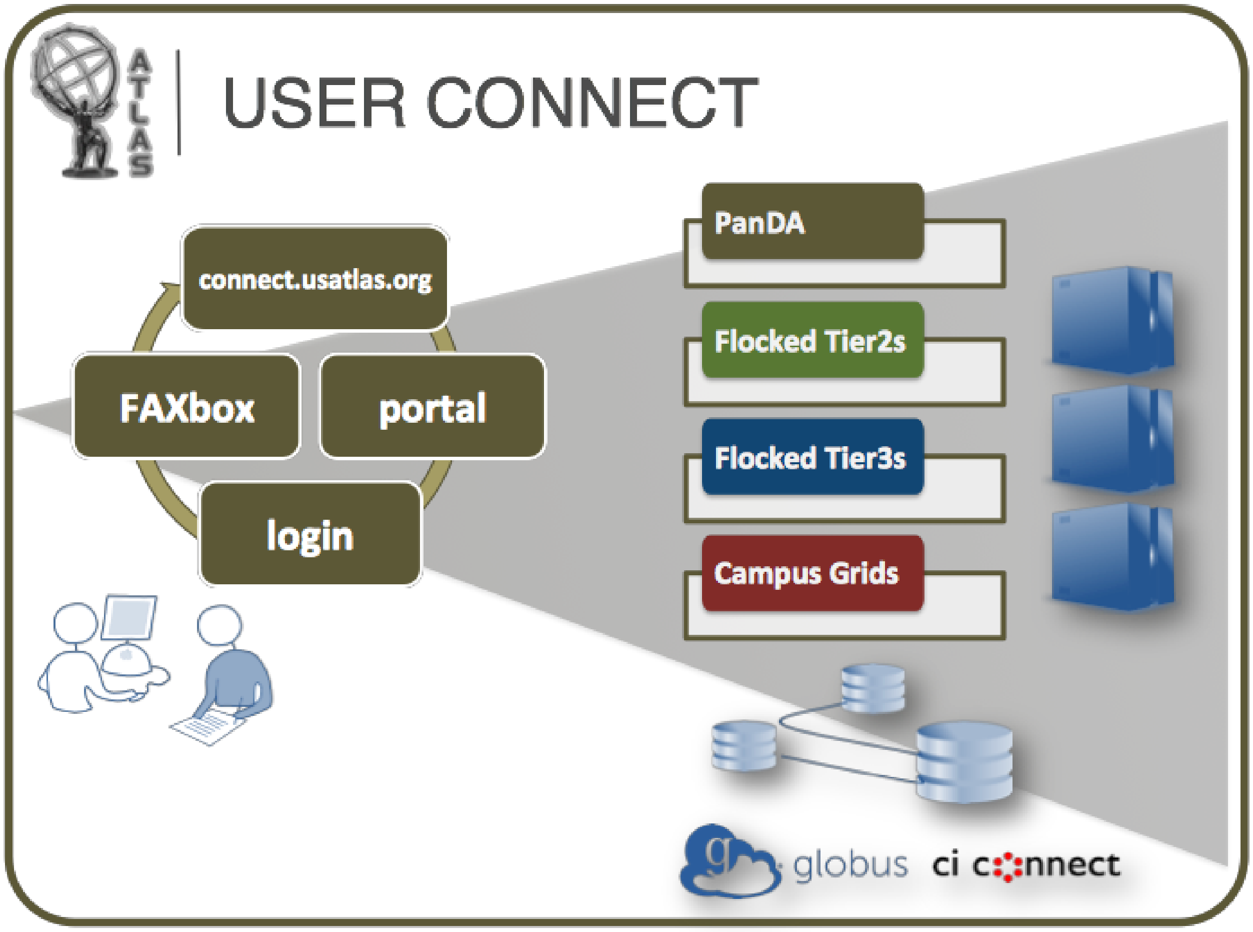
ATLAS CONNECT REQUIREMENTS SUMMARY

- Provide services which allows existing Tier3 clusters to "connect" to the fabric of Tier2 and Tier1 clusters.
- An interactive login service accesible by anyone in US ATLAS
- Connection to a federated identity management system that supports using local campus identity for user authentication
- An authorization service to permit user access to distributed resources accessed by the system
- An HTCondor job submission service to connected resources
- A service to facilitate Panda job management capabilities (e.g. pilot factory submission)
- A locally mounted scratch space with significant capacity to support analysis processing at a per-user scale to be determined by the US ATLAS Resource Allocation Committee
- Allow sharing of private resources between institutions and users, and the US

ATLAS-wide Connect service.

- As the resource provisioning layer improves, allow the ability to burst into cloud resourcs (Amazon, Google, etc.) automatically.

ATLAS CONNECT USER



Initial Components

- connect.usatlas.org, webpage describing the service and point of entry for sign up/sign in, accounting and monitoring displays, and links to documentation and support. **Deployed**.
- portal: service for managing group membership and authorizing new users. **Deployed.**
- login: a hosted submit host(s) service with HTCondor, AutoPyFactory and PanDA job submit capabilities to a variety of targets, both within and beyond dedicated facilities. **HTCondor capability deployed**.
- FaxBox: a Posix scratch data service to store job outputs, accessible locally via POSIX file operations (cp, rm, mv, mkdir, etc) and remotely via Xrootd, http, gridftp, and Globus. It will **not** be part of DDM, and therefore, invisible to ADC operations. **Basic Ceph**

filesystem deployed, testing.

- For institutions with CI Connect campus grids (such as Duke University CI Connect), FaxBox as well as institutional scratch in the Stash data service, local mount points are provided so that /home and /data scratch areas can be shared between the two CI Connect platforms. Duke CI Connect currently bridges the Duke University campus grid offerred by the SCC (Duke Grid), UC2 (University of Chicago Computing Cooperative), and the OSG. **Deployed**.

ATLAS CONNECT SERVICES

Below we list details about requirements and in some cases descriptions of components already implemented and deployed.

Identity Management Service

- We leverage the Nexus identity management service as implemented by Globus. Details can be found here.
- Allows use of native campus identity management systems, CILogin and the InCommon federation. Essentially, a user can signup to ATLAS Connect using his institution network ID and password.
- Also provides identity, profile and group management service.
- All this is **deployed** leveraging CI Connect (portal.ci-connect.net). This will be replaced with a “branded” version specific to ATLAS Connect. While the functionality will be unchanged, this would insure a more uniform look-and-feel for ATLAS users.

Automatic Account Provisioning Service

- A host, **login.usatlas.org** which provides a user login with a /home directory, sym link to user storage on FAXbox (/home/user/data) is automatically provisioned once a user is authorized to the system. A user is authorized through a process similar to identity verification for grid certificates (i.e. via a certified Registration Agent). The technology for this was adapted from the U-Bolt toolkit developed at the University of Chicago. **Deployed.**

Job Submission Service

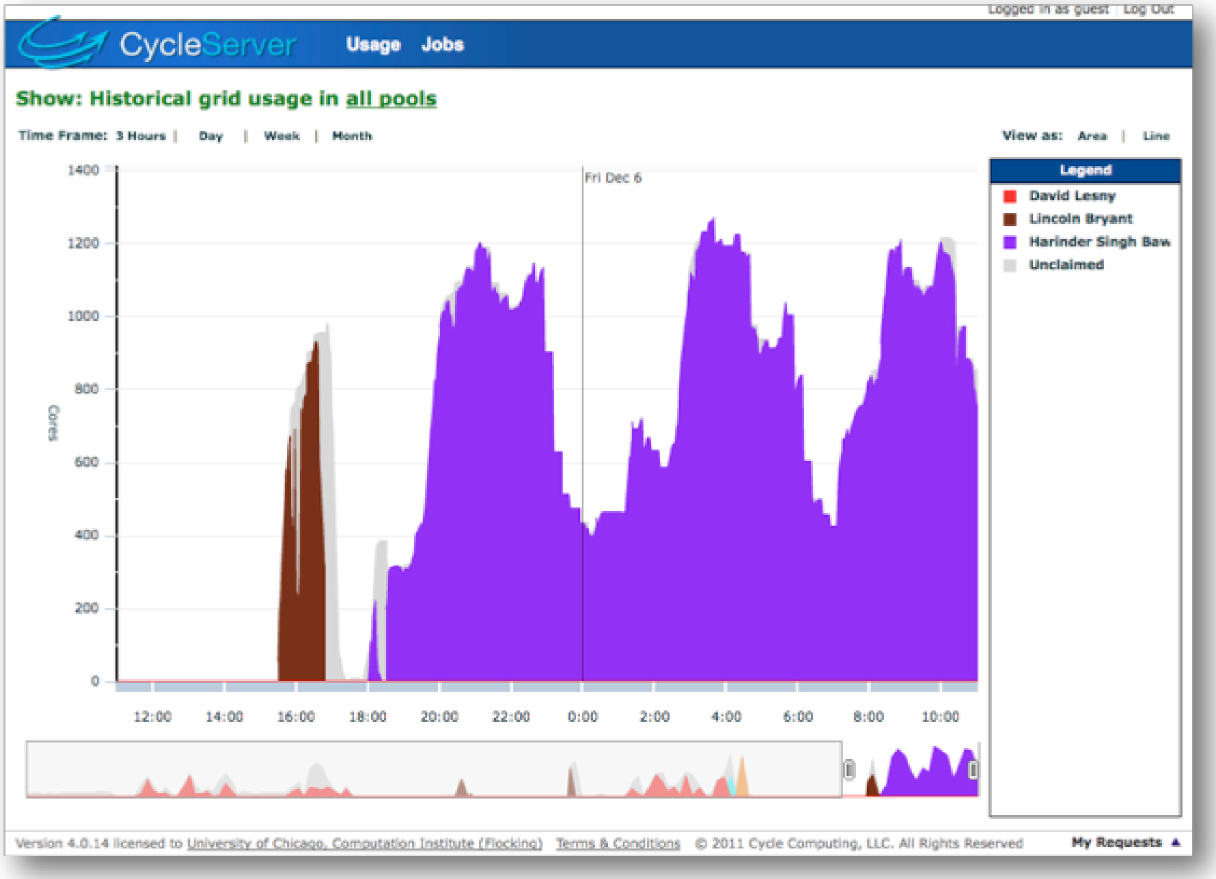
- The login host provides an HTCondor schedd service configured to directly flock to various targets. Currently, these include CSU Fresno, AGLT2, MWT2, and UC3 (UChicago campus grid). The ATLAS Connect resources are monitored using a liscense donated by Cycle Computing - see below. The monitor also displays usage by user name. Basic job throughput testing is being done by CSU Fresno (Harinder Bawa). **Deployed.**

ATLAS Connect XSEDE Science Gateway (TACC Stampede)

- At the Arizona Facilities meeting Rob, Michael and Peter Onyisi (new faculty hire at UTexas) discussed mechanisms for accessing Stampede, a signficant computing resource at TACC.
- The essential idea would be to create an RCCF instance (a BOSCO-based factory service, see below) which uses a group account and submits simple HTCondor jobs into the local (SLURM) Stampede scheduler.
- As TACC is an XSEDE resource, we were advised to follow the Science Gateway model https://www.xsede.org/gateways-overview, and will begin discussions with Nancy Wilkins-Diehr from SDSC who oversees the program in January. This will likely require a formal allocation process at some point, and a gateway account. However using Peter's UT faculty status we can easily acquire a startup allocation.
- We cannot mount the ATLAS CVMFS repository on TACC resources. However, Parrot (a system which traps system IO calls, see http://www3.nd.edu/~ccl/software/parrot/) can be used to mount remote file systems. For the OSG Connect tutorial at the Duke workshop we demonstrated this for simple ROOT applications run from anywhere.
- Peter is working with the Notre Dame CC Tools team to fix issues Parrot has with Athena. Once those are overcome, a broader scope of applications will be possible.
- FAX and FAXbox will be used for data access and job output, in the Tier3-like context.
- Panda pilots can be submitted to reach TACC allocations as well, as discussed below. In that case we could minimize the local effort required to provide the resource by using Autopyfactory, Bosco, and the MWT2 storage endpoints.

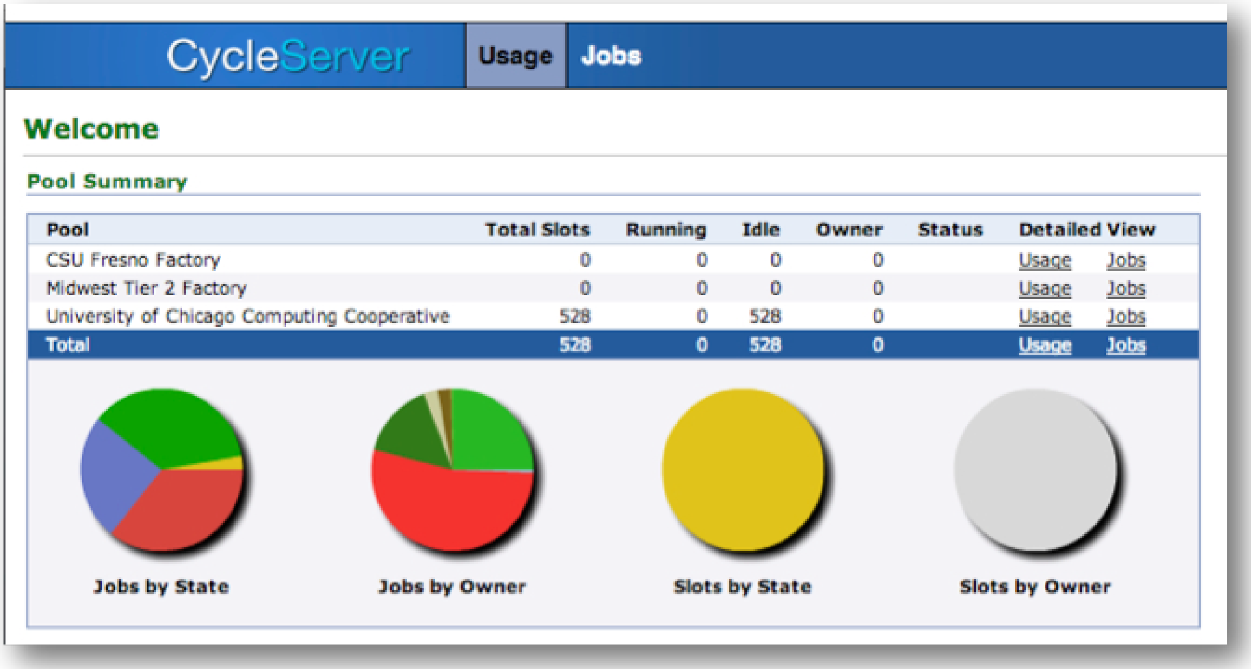
Flocking Service to OSG

- ATLAS Connect has the ability to reach opportunistic resources of OSG by flocking to the OSG VO-XD “front-end” machine. While ATLAS’ CVMFS software repository is not gauranteed to be present on all OSG sites, Parrot can be used access CVMFS for simple analysis jobs as discussed above.



Show: Historical grid usage in all pools

Version 4.0.14 licensed to University of Chicago, Computation Institute (Flocking) Terms & Conditions © 2011 Cycle Computing, LLC. All Rights Reserved

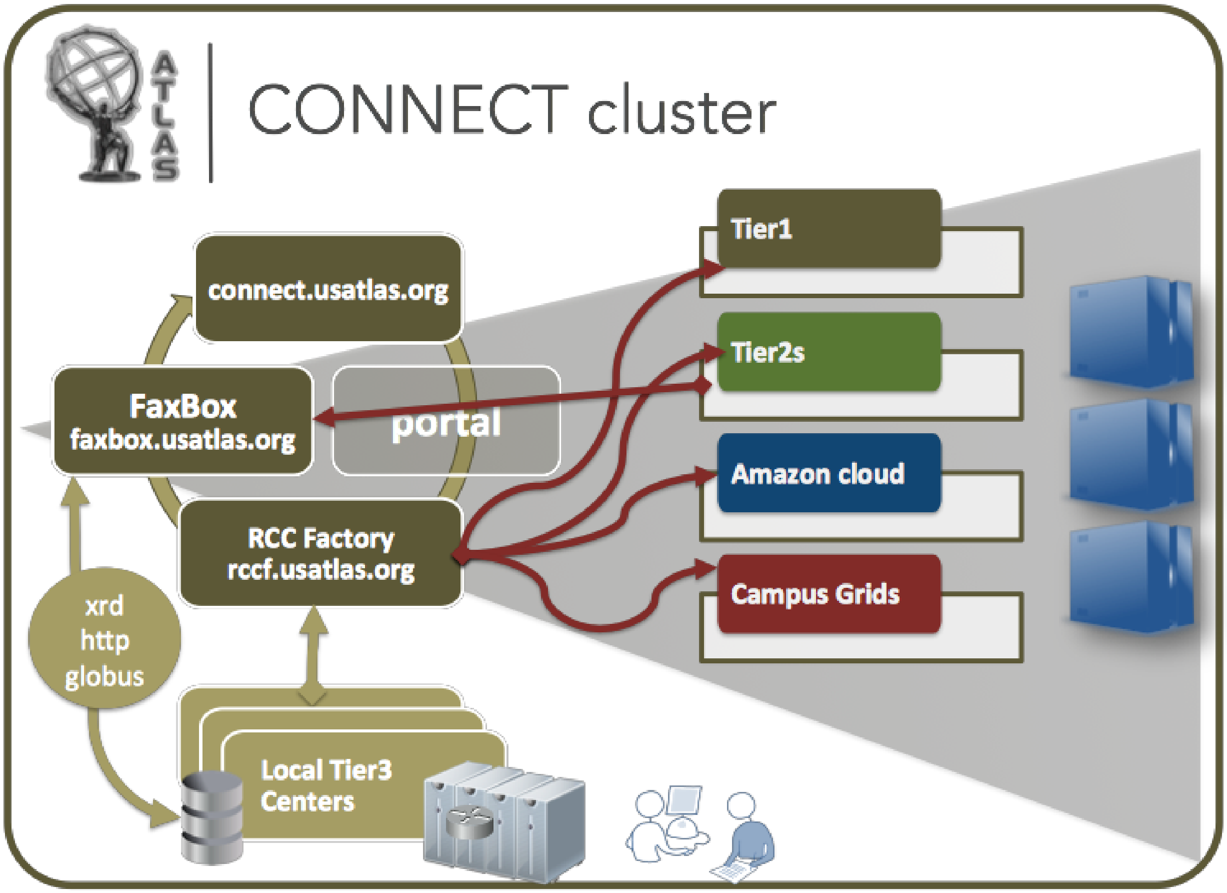


Welcome

Pool Summary

Pool	Total Slots	Running	Idle	Owner	Status	Detailed View	
CSU Fresno Factory	0	0	0	0		Usage	Jobs
Midwest Tier 2 Factory	0	0	0	0		Usage	Jobs
University of Chicago Computing Cooperative	528	0	528	0		Usage	Jobs
Total	528	0	528	0		Usage	Jobs

ATLAS CONNECT CLUSTER



The Remote Cluster Connection Service (RCCF - the RCC Factory)

- Purpose: attach a Tier 3 cluster to the fabric of Tier 2 centers.
- Use HTCondor flocking technology, and a campus pilot factory service running at a Tier 2 center. **This has been deployed and is functional: rccf.usatlas.org with resource targets at MWT2 and AGLT2.**
- Flocked Tier3s sites: the Argonne Tier3 test cluster, IU, UC, UIUC are functional and have been tested. Factories have been setup for Argonne, Arizona, Duke, and UC Irvine.
- Basic workflow: a user works entirely at his home Tier3. For Tier3's using Condor as the job manager, the submit host queue can be configured to 'overflow' jobs to the connected resources, e.g. beyond-pledge Tier2s. For non-Condor Tier3 clusters, a Condor queue can be installed locally (i.e. a Condor submit host).
- User documentation with demonstrated Tier3 → Tier2 analysis examples is needed. The Tier3 flocking setup and basic job submission documentation is available here:

- http://twiki.mwt2.org/bin/view/Main/LocalClusterSetupToFlockIntoRCCF

RCCF Technical details

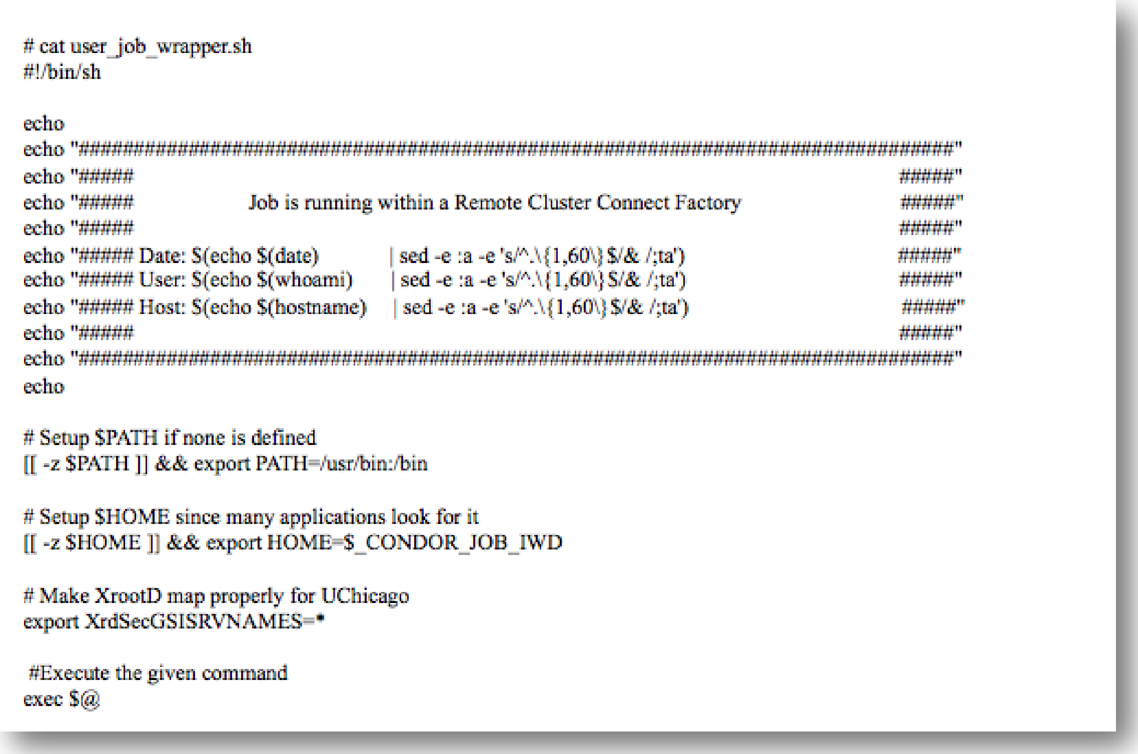
- Each RCCF is a separate Condor pool with a SCHEDD/Collector/Negotiator
- The RCCF injects glideins via SSH into target SCHEDDs at MWT2 and AGLT2
- The glidein creates a virtual job slot from Target SCHEDD to the RUCF
- Any jobs which are in that RCCF then run on either the MWT2 or AGLT2 Condor pools
- Jobs are submitted to the RCCF by flocking from a source SCHEDD
- The RCCF can accept flocked jobs from multiple source SCHEDD hosts; for example, from the SCHEDD running on **login.usatlas.org.**
- Must have open bidirectional access to at least one port on the target SCHEDD
- Firewalls can create problems – SHARED_PORT makes it easier (single port)
- The system has been Puppetized using the MWT2's configuration management system so that new factories (for new targets) can be easily built.
    - `bosco_factory` – Create a RCC Factory
        - Define the user account and shared port factory runs in
        - Other parameters to change max glideins, max running, etc
        - User account must exist on uct2-bosco (puppet rule)
        - Installs bosco, modifies some files, copies host certificate
    - `bosco_cluster` – Create a Bosco Cluster to a target SCHEDD
        - Creates Bosco Cluster to target SCHEDD
        - User account must exist at target and have SSH keys access
        - User account can be anything Target SCHEDD admin allows
        - Pushes job wrapper, condor_submit_attributes, etc
    - `bosco_flock` – Allow a source SCHEDD to flock to this Factory
        - Source SCHEDD FDQN
        - For GSI – DN of the Source SCHEDD node
    - `bosco_require` – Add a "requirement" (ClassAd) to a slot
        - Allows one to add a classAD to a slot, ror example - HAS_CVMFS
        - Two classADs added to a factory by default
            - `IS_RCC = True`
            - `IS_RCC_<factory nickname> = True`
        - Remote Users can use these in their Condor submit file
    - Example Puppet rule for a Tier3→Tier2 flocking factory:

```
bosco::factory { 'uiuc'    : bosco_factory             => 'uiuc',
                             bosco_port                => '11010',
                             bosco_maxrunningjobs      => '1000',
                             bosco_maxidleglideins     => '25',
                             bosco_iterationtime       => '15',
                             bosco_maxqueuedjobs       => '25',
                             bosco_version            => '1.2',
                             bosco_security           => 'gsi'
}
```

- Sample Tier3 HTCondor configuration

  ```
  # Setup the FLOCK_TO the RCC Factory

  FLOCK_TO = $(FLOCK_TO),
  rccf.usatlas.org:<RCC_Factory_Port>?sock=collector

  # Allow the RUC Factory server access to our SCHEDD

  ALLOW_NEGOTIATOR_SCHEDD = $(CONDOR_HOST), rccf.usatlas.org

  # Who do you trust?

  GSI_DAEMON_NAME = $(GSI_DAEMON_NAME), /DC=com/DC=DigiCert-Grid/O=Open
  Science Grid/OU=Services/CN=rccf.usatlas.org

  GSI_DAEMON_CERT = /etc/grid-security/hostcert.pem

  GSI_DAEMON_KEY = /etc/grid-security/hostkey.pem

  GSI_DAEMON_TRUSTED_CA_DIR = /etc/grid-security/certificates

  # Enable authentication from tne Negotiator (This is required to run on
  glidein jobs)

  SEC_ENABLE_MATCH_PASSWORD_AUTHENTICATION = TRUE
  ```

- Job Wrapper Script
  - Setup a minimum familiar environment for the user
  - Print a job header to help us know when and where the job ran
  - Date, User and hostname the job is running on
  - Define some needed environment variables:
    - `$PATH` – System paths (should we add /usr/local, etc)
    - `$HOME` – Needed by ROOT and others
    - `$XrdSecGSISRVNAME` – Works around a naming bug
    - `$IS_RCC=True`
    - `$IS_RCC_<factory>=True`
  - Exec the user executable
  - Other options:
    - AtlasLocalRootBase (probably not as the user can do this)
    - Access to FAXbox
    - Access to LOCALGROUPDISK
  - The script:



```
# cat user_job_wrapper.sh
#!/bin/sh

echo
echo "############################################################################################"
echo "######                                                                                ######"
echo "######                   Job is running within a Remote Cluster Connect Factory       ######"
echo "######                                                                                ######"
echo "###### Date: $(echo $(date)          | sed -e :a -e 's/^.\{1,60\}$/& /;ta')              ######"
echo "###### User: $(echo $(whoami)        | sed -e :a -e 's/^.\{1,60\}$/& /;ta')              ######"
echo "###### Host: $(echo $(hostname)      | sed -e :a -e 's/^.\{1,60\}$/& /;ta')              ######"
echo "######                                                                                ######"
echo "############################################################################################"
echo

# Setup $PATH if none is defined
[[ -z $PATH ]] && export PATH=/usr/bin:/bin

# Setup $HOME since many applications look for it
[[ -z $HOME ]] && export HOME=$_CONDOR_JOB_IWD

# Make XrootD map properly for UChicago
export XrdSecGSISRVNAMES=*

 #Execute the given command
exec $@
```

- HTCondor submit file example
- Job Accounting and ATLAS Connect Cluster **work in progress**
  - Gratia records from flocked jobs are not associated with a VO
  - Simple addition to the gratia config, maps these into the Atlas VO
  - Addition made to /etc/gratia/condor/ProbeConfig
    - MapUnknownToGroup="1"
    - MapGroupToRole="1"
    - VOOverride="atlas"
- Might enable gratia on rccf.usatlas.org for each factory instance

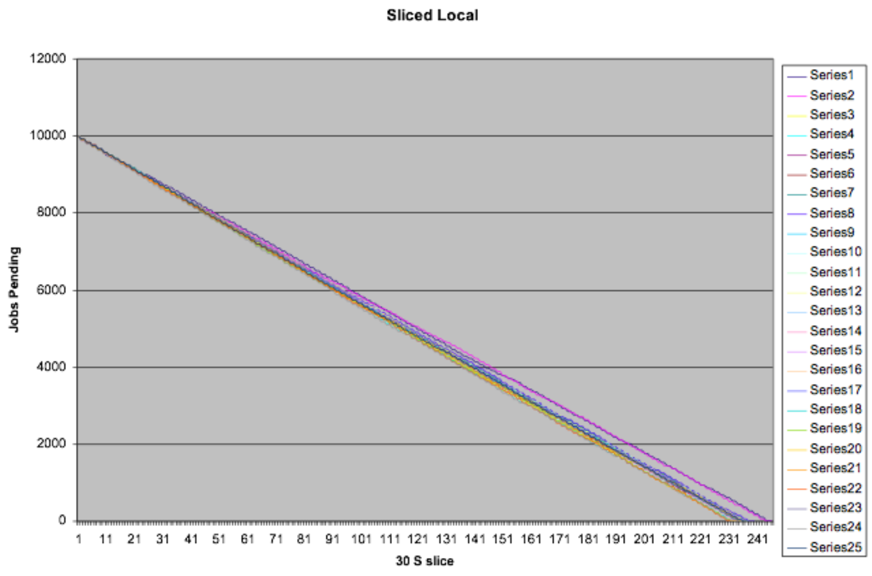
Performance

We have made a few performance measurements for jobs submitted to Tier3 clusters wich flocked to ATLAS Cluster Connect. In this case the jobs ran on MWT2. Some notes:

- Jobs will run the same no matter how they arrive on an MWT2 worker node
- Submission rates (Condor submit to Execution) are the key
- Local submission involves only local SCHEDD/Negotiator/Collector
- Flocking from Tier3 → Tier2 has multiple steps
  a. Local submission with SCHEDD and Negotiator
  b. Local SCHEDD contacts RCCF Factory Negotiator
  c. RCCF Negotiator matches jobs to itself and they flock

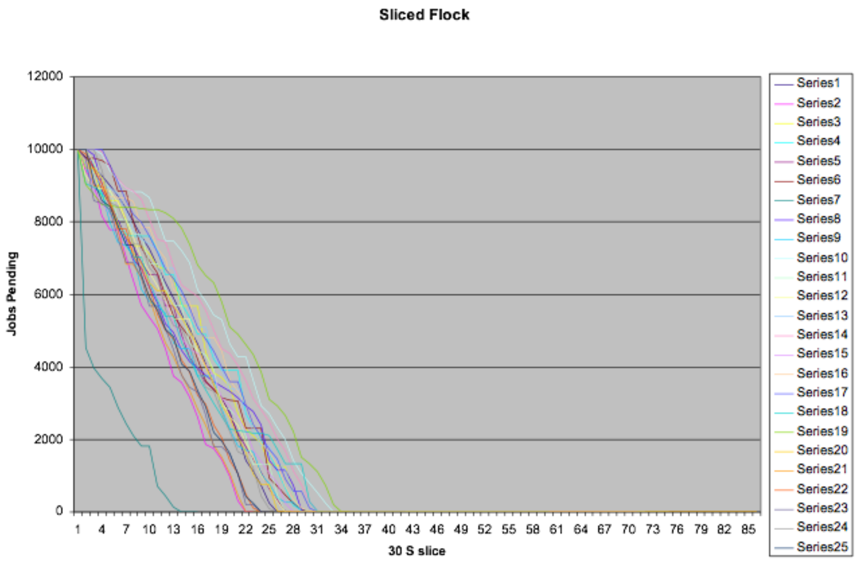
    d. The RCCF factory SSH into an MWT2 SCHEDD and creates a virtual slot on the MWT2
    e. Job begins execution in a free virtual slot on the MWT2 worker node

- To test submission rates, the earlier given simple submission was used
    a. Submit 10000 jobs to both the local Condor pool and RCC Factory
    b. Start the clock after the “condor_submit” with a “Queue 10000”
    c. Loop checking when all jobs have completed with “condor_q”
    d. Jobs are only “/bin/hostname” so they exit almost immediately
    e. Wall clock time between start and end will be a 10K submission rate
    f. Difference between local and RCCF test should show the overhead
    g. Local rate dependent on number of local jobs slots
    h. Negotiator cycle time (60 seconds) also plays a big role
    i. Local submission rates depend on number of job slots and Negotiator rate
        - Used LX cluster at Illinois
        - 62 empty job slots
        - Default Negotiator cycle (60 seconds)
        - Use 10K batch submissions to remove bias of a small sample
        - Value under 60 can happen within seconds to just over 60
        - Results follow, displayed as #pending jobs versus wall time



    j. Remote test dependent on how quickly slots become available
        - Ran 25 tests, 30 second samples
        - Results follow, displayed as #pending jobs versus wall time for the flocked mode.
        - In this comparison, the flocked jobs running on the much larger resource pool which had, consequently, more available slots, finished an order of magnitude more quickly: ~25 seconds (flocked) versus 240 seconds

(local). This is a measure of how quickly resources can be harnessed and completed, compared local (quicker, but more limited) resources.



ATLAS CONNECT PANDA

The goal is to provide a bridge between Panda job management services and ATLAS Connect resources. This would resemble the long-discussed “US Queue”. The real value added for this approach is to extend the reach of Panda to off-grid locations on campus clusters which don’t have an OSG compute element. **To be deployed**.

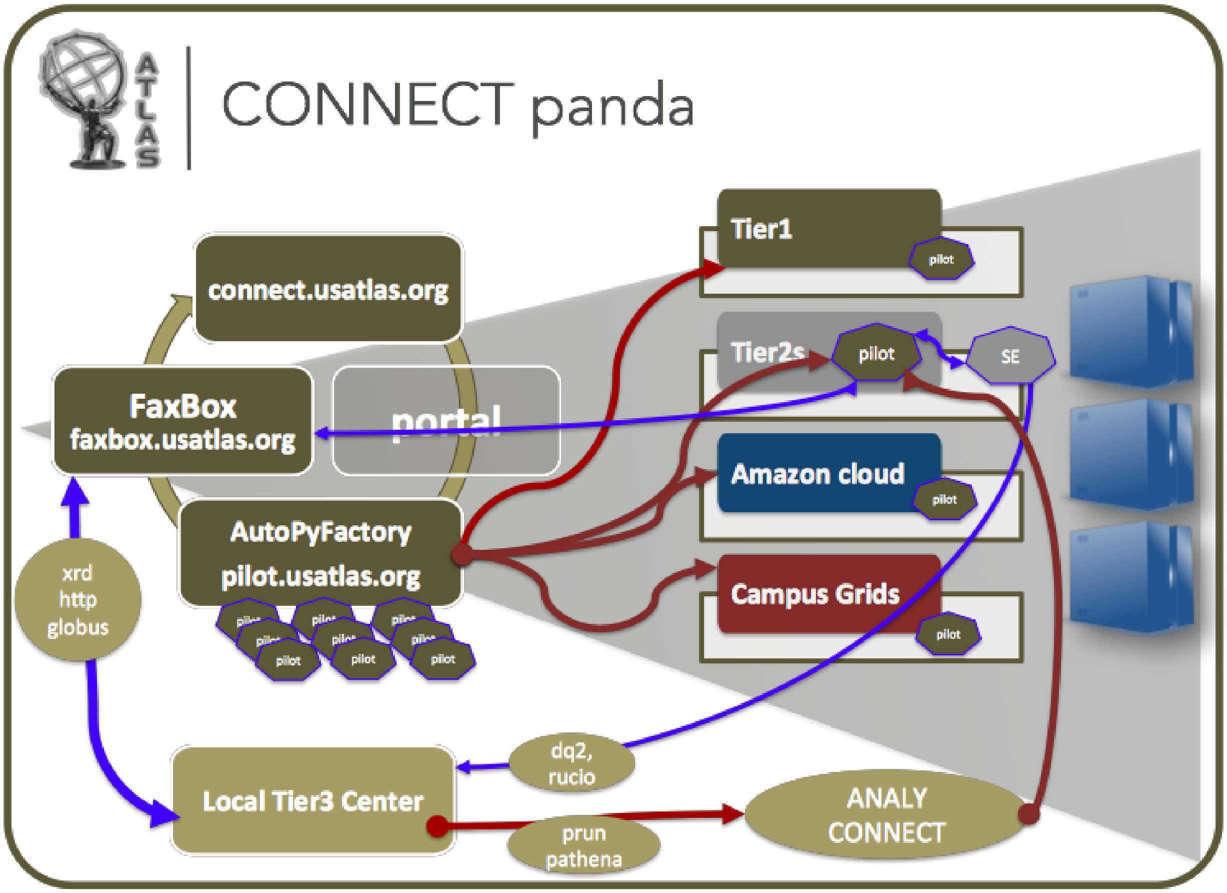
Components

- AutoPyFactory service
- ANALY_CONNECT_US - an analysis queue
- CONNECT_US_PROD - a production queue
- MWT2 Storage Element (to associate with the queues)
- Resources: initially BNL, MWT2, AGLT2 (for beyond-pledge analysis processing while testing the system).
- 12/31/2013: Stampede at TACC is becoming available with a startup allocation already granted. Parrot is used to mount CVMFS. Peter has finished a test using Parrot and a full reconstruction from RAW data. Autopyfactory will be used to submit Panda pilots to a Stampede (BOSCO) submit factory, similar to T3 factories. More on this below.

Usage

- AutoPyFactory submits pilots across multiple ATLAS Connect resources, choosing only pilot-capable resources which meet the minimum requirements (e.g. HAS_CVMFS).

- Users submit jobs as usual from their Tier3, or wherever, using prun or pathena, selecting ANALY_CONNECT_US.
- Pilots communicate to the ANALY_CONNECT_US queue in the normal way; the MWT2 SE can be used as the output storage element.



AUTOPYFACTORY (APF)

Local Pilot Factory

The essential purpose for the APF is to provide a local source of ATLAS pilots thus avoiding the need for a full-fledge grid compute element.   Note in this instance APF will submit pilots using multiple sites using HTCondor as opposed to (grid-based) HTCondor-G.

Instructions from John Hover

The page in the ATLAS Twiki:
https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/AutoPyFactory

But primarily you want to follow the documentation in the distribution:

http://svnweb.cern.ch/guest/panda/panda-autopyfactory/current/INSTALL-ROOT

Point at the testing repo, and you'll be installing

```
panda-autopyfactory
panda-autopyfactory-tools
wrapper
```

(use version 0.9.10 within the RPM--it contains more than one)

Suitable wrapper argument configuration is already in the sample queues.conf-example file. I just added a Condor Local example to the queues.conf-example file in SVN:

http://svnweb.cern.ch/guest/panda/panda-autopyfactory/current/etc/queues.conf-example