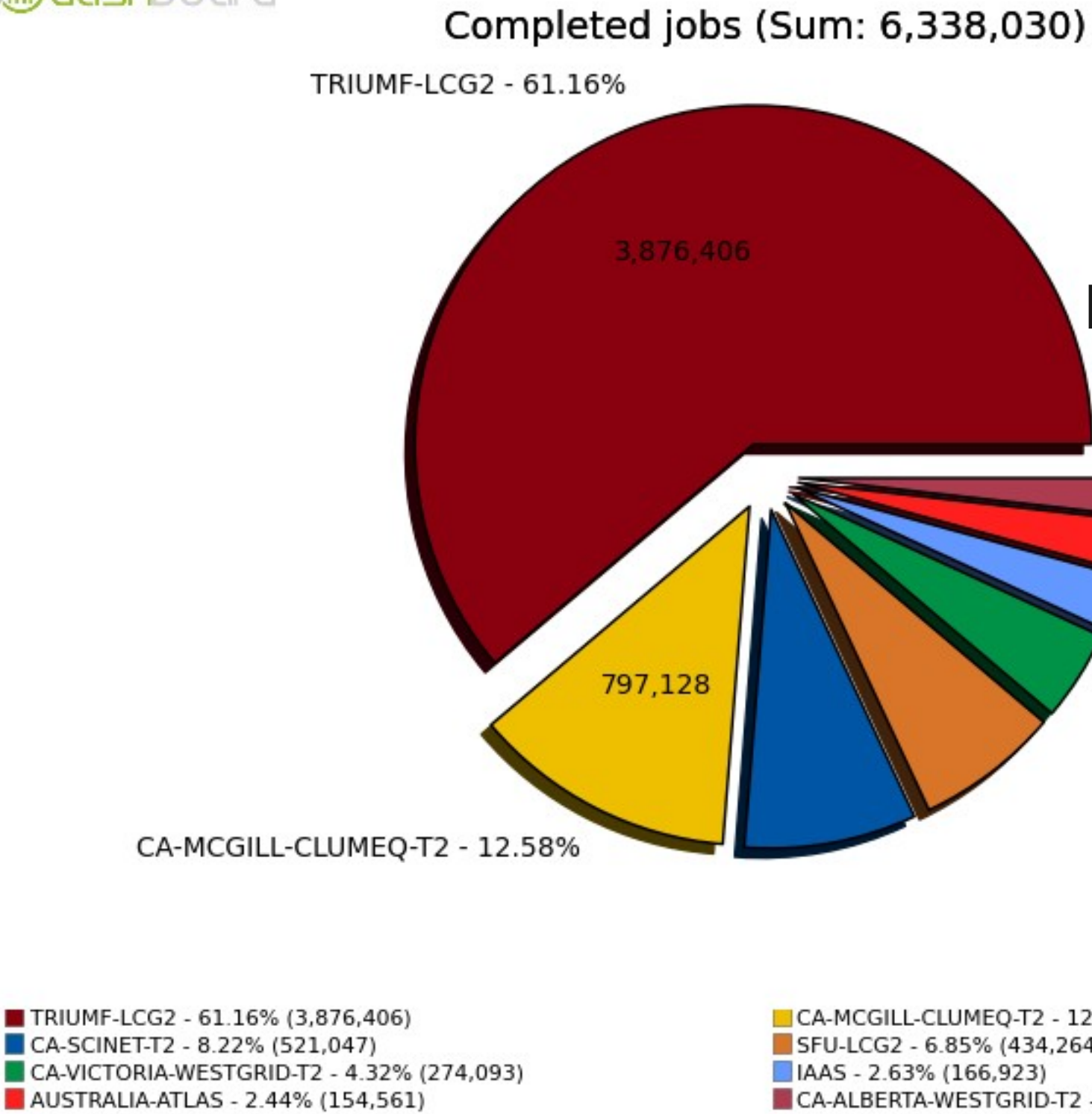
Providing IaaS Resources to ATLAS: The UVic-NeCTAR Experience

Ashok Agarwal, Andre Charbonneau, Asoka de Silva, Ian Gable, Joanna Huang, Colin Leavett-Brown, Michael Paterson, Randall Sobie, Ryan Taylor

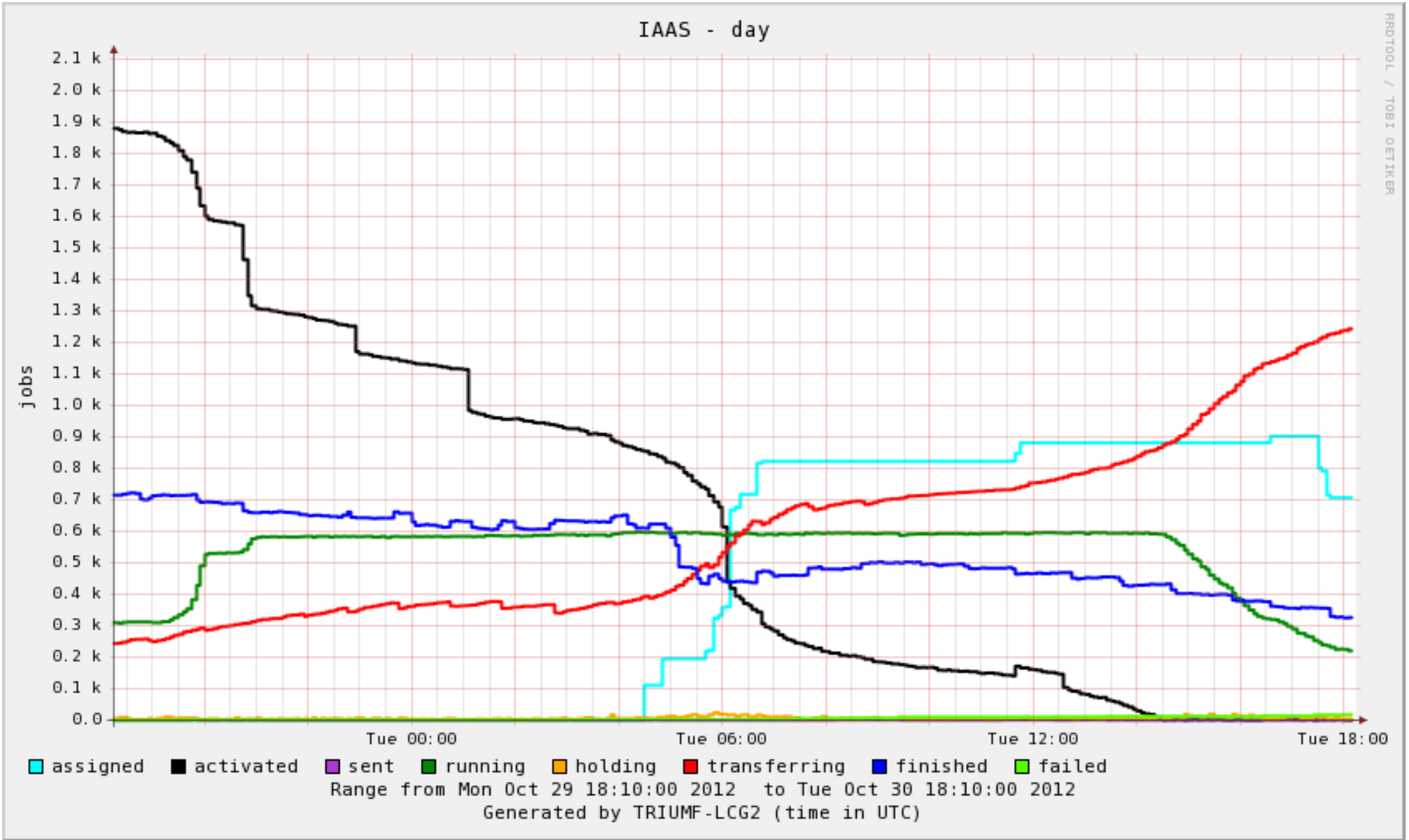
CA Cloud Production Activity, Last 7 Months



IAAS

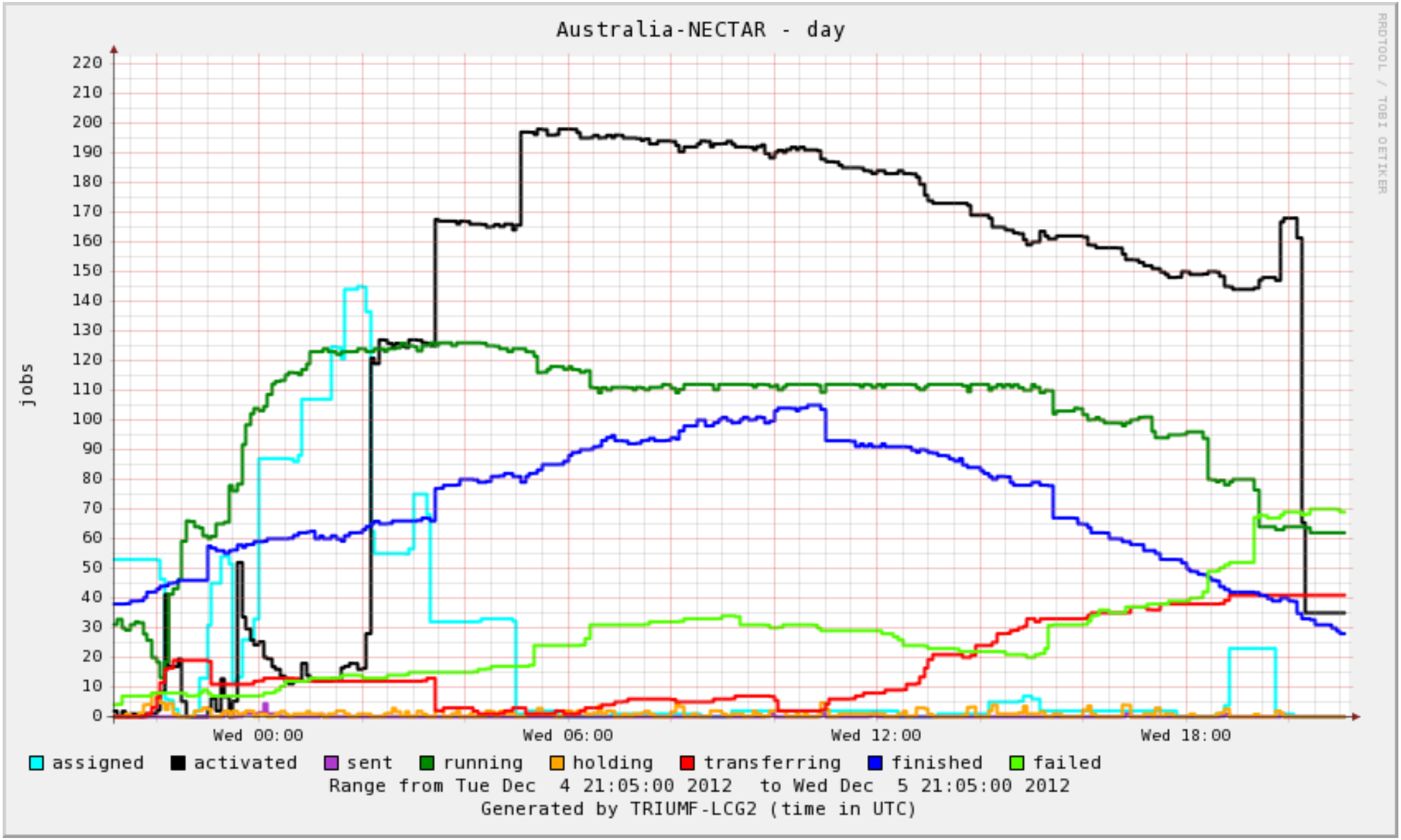
IAAS

- Early tests Nov. 2011, standard operation since April 2012



Australia-NECTAR

- Commissioned Dec. 2012, still in early stages

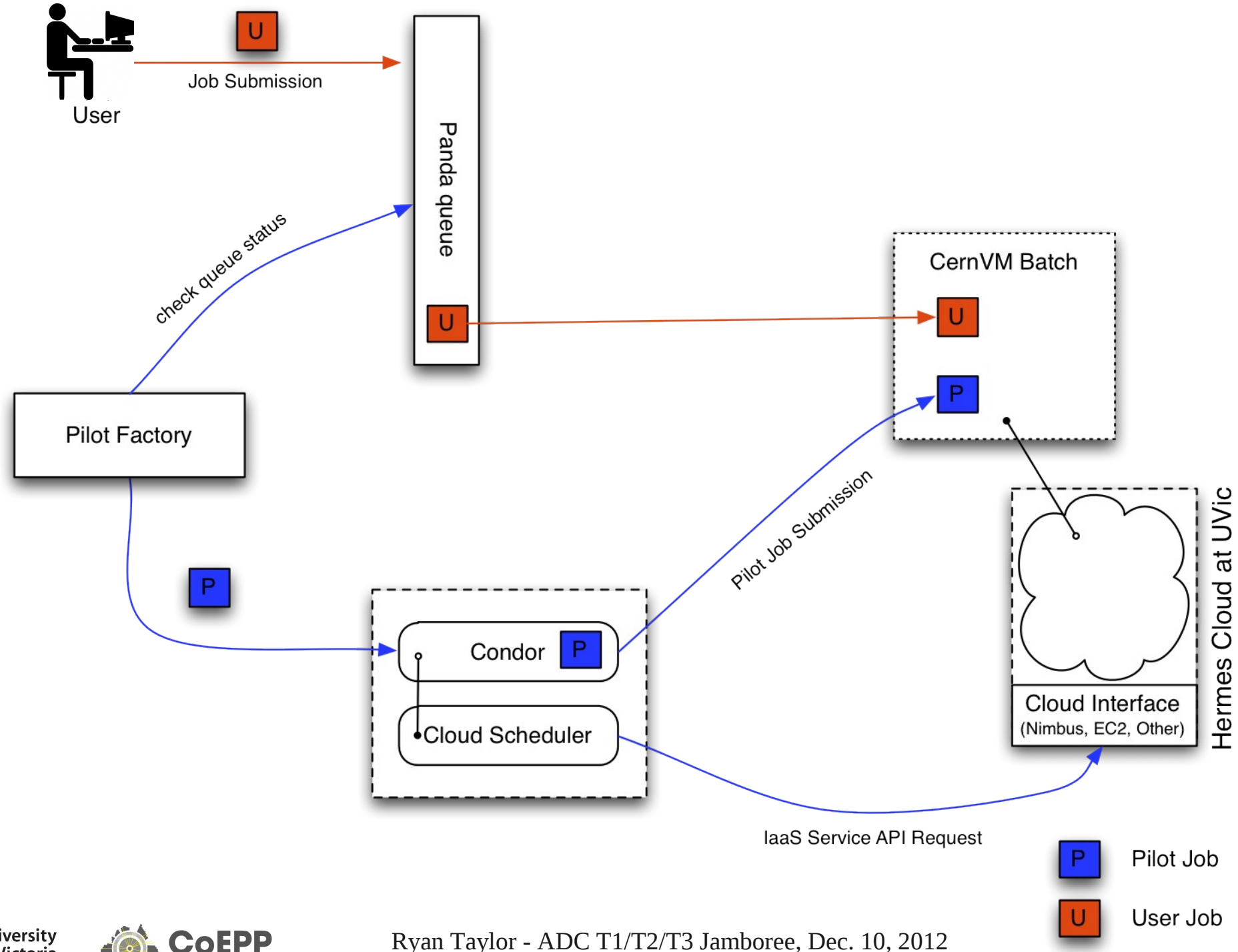


Powered by Cloud Scheduler

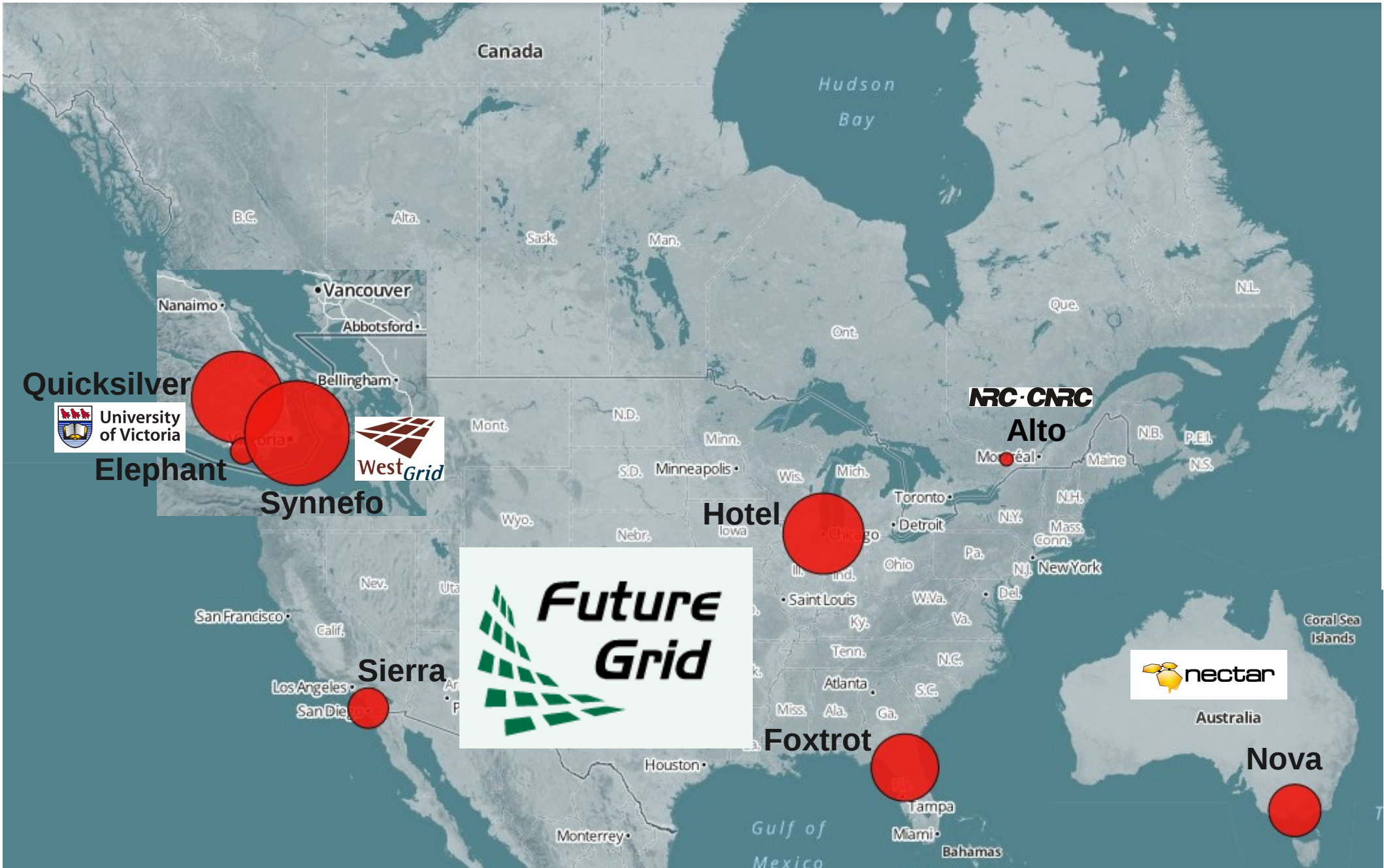
- Cloud Scheduler is a simple python package for managing VMs on IaaS clouds, based on the requirements of Condor jobs
- Users submit Condor jobs, with additional attributes specifying VM properties
- Developed at UVic and NRC since 2009
- Used by BaBar, CANFAR, ATLAS
- http://cloudscheduler.org/
- http://goo.gl/G91RA (ADC Cloud Computing Workshop, May 2011)
- http://arxiv.org/abs/1007.0050

Key Features of Cloud Scheduler

- securely delegates user credentials to VMs, and authenticates VMs joining the Condor pool.
- interacts with multiple IaaS sites, and aggregates their resources under one Condor queue.
- dynamically manages quantity and type of VMs in response to user demand.


User
U
Job Submission
Panda queue
U
check queue status
CernVM Batch
U
P
Pilot Factory
P
Condor
P
Cloud Scheduler
Pilot Job Submission
Cloud Interface
(Nimbus, EC2, Other)
Hermes Cloud at UVic
IaaS Service API Request
P
Pilot Job
U
User Job

Participating Clouds



VM Image

- Dual-hypervisor image, can run on KVM or Xen
- Customized CernVM Software Appliance batch node v2.6.0
- Use whole-node VMs for better efficiency
  - cache sharing instead of disk contention
  - fewer image downloads when ramping up

Data Access

- IAAS and Australia-NECTAR are linked to their T2 SEs
- Our approach has been to dynamically create compute resources, with remote access to static storage outside the cloud
- Satisfactory for now
  - MC production is low I/O, ideal use-case
- But not scalable long-term
  - Eventually should use a storage federation

Adding IaaS Resources to The “Grid of Clouds”

- Step 0 - Get an IaaS cloud
- Step 1 - Boot VMs
- Step 2 (optional) - Get a Panda queue
- Step 3 (optional) - Run your own Cloud Scheduler

Step 0: Get An IaaS Cloud

- Cloud Scheduler supports:
  - Nimbus
  - Amazon EC2
  - OpenStack
  - StratusLab
  - OpenNebula
- Then, use your cloud!

Step 1: Boot VMs

- Allow Cloud Scheduler server to boot VMs
  - Analogous to allowing a DN to submit grid jobs to a CE
- Test the image (may need customization)
  - We can provide an image to use
- Run some VMs, join condor pool
- Then, run condor jobs!
  - If joining an existing Panda queue, you're already done!

Optional Step 2: Get a Panda Queue

- Make a Panda site, with prod and analy queues
- Associate with a SE
- Use WAN protocol (e.g. lcgcp, curl) for stagein
- Enable AFT/PFT jobs in HammerCloud, and switcher for downtimes
- Create site in AGIS (but not GOCDB)
- Then, run Panda jobs!

Optional Step 3: Run Your Own Cloud Scheduler

- For a fully independent and complete solution
  - Install condor server
  - `pip install cloud-scheduler`
- Maybe even your own Pilot Factory

Missing Pieces

- APEL accounting in the cloud
- Ability to declare downtime on a Cloud Scheduler server
- SW release publication in AGIS without a CE

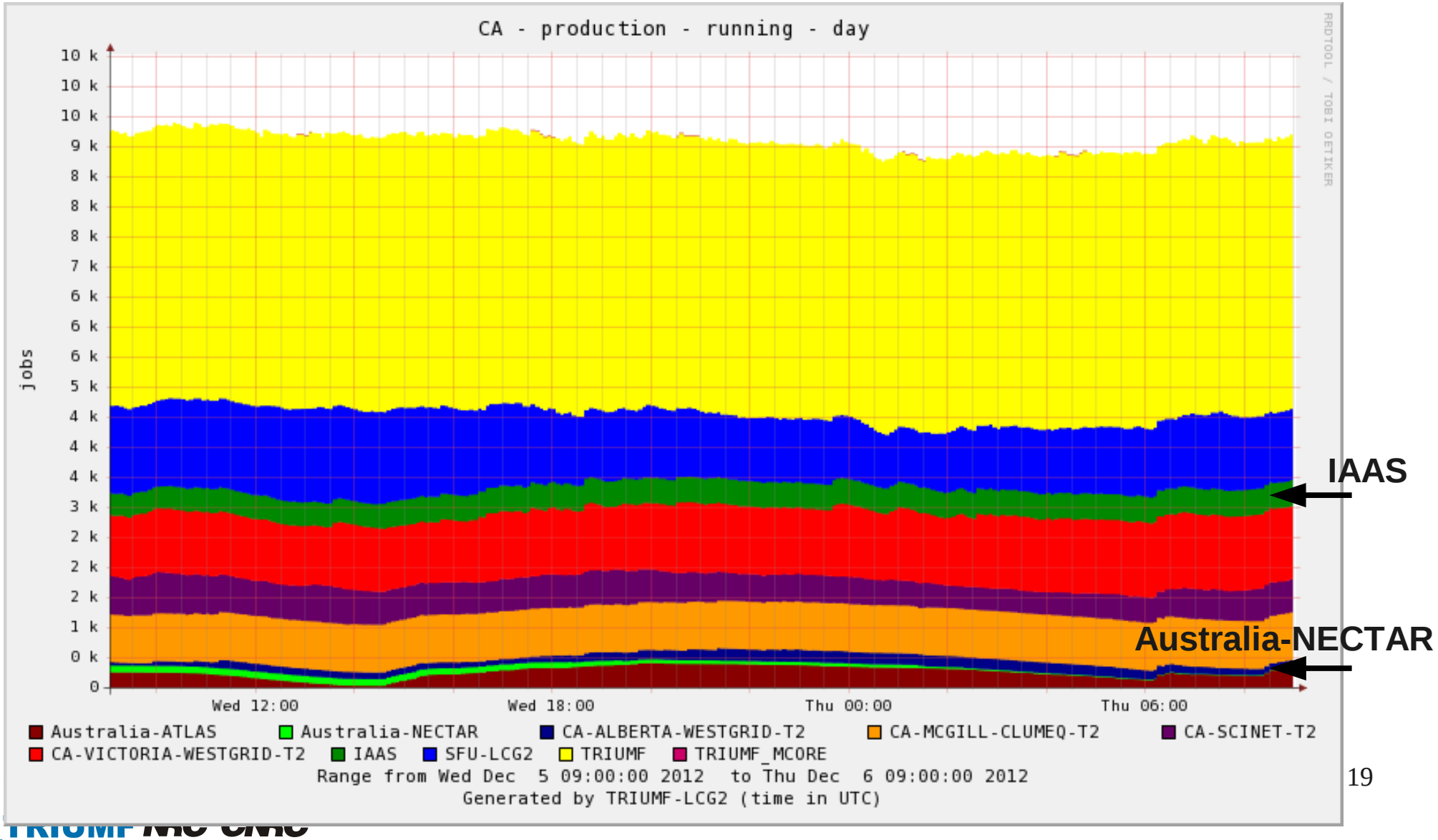
Conclusion

- Developed and deployed an infrastructure to transparently run jobs in Panda queues spanning multiple IaaS clouds
- Using it to deliver beyond-pledge resources to ATLAS
  - In IAAS, completed 177K prod jobs since April
- Recently created the Australia-NECTAR cloud site running on another continent

Extra Material

CA Production Queues

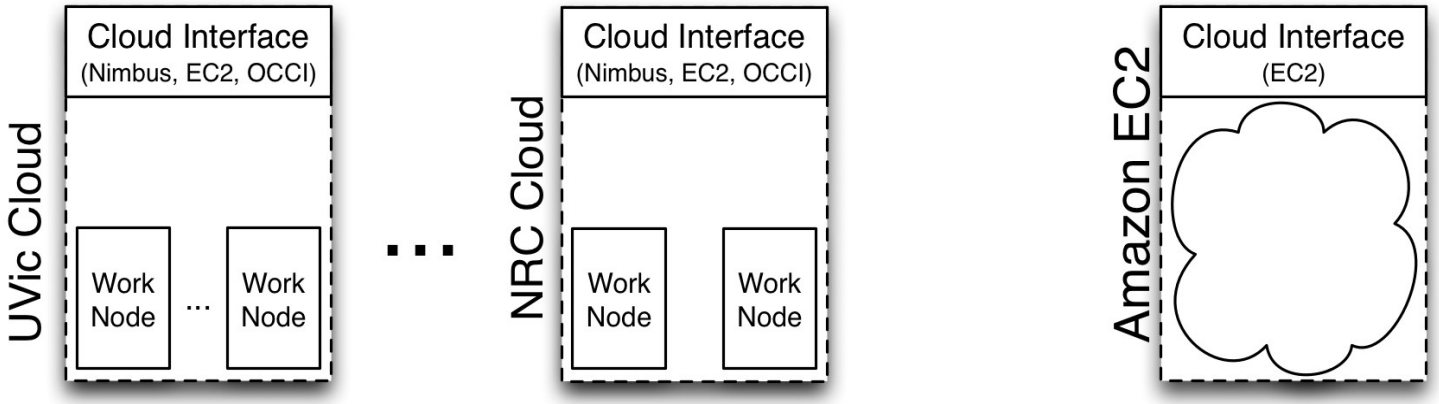
- Two are in the cloud: IAAS and Australia-NECTAR



Condor Job Description File

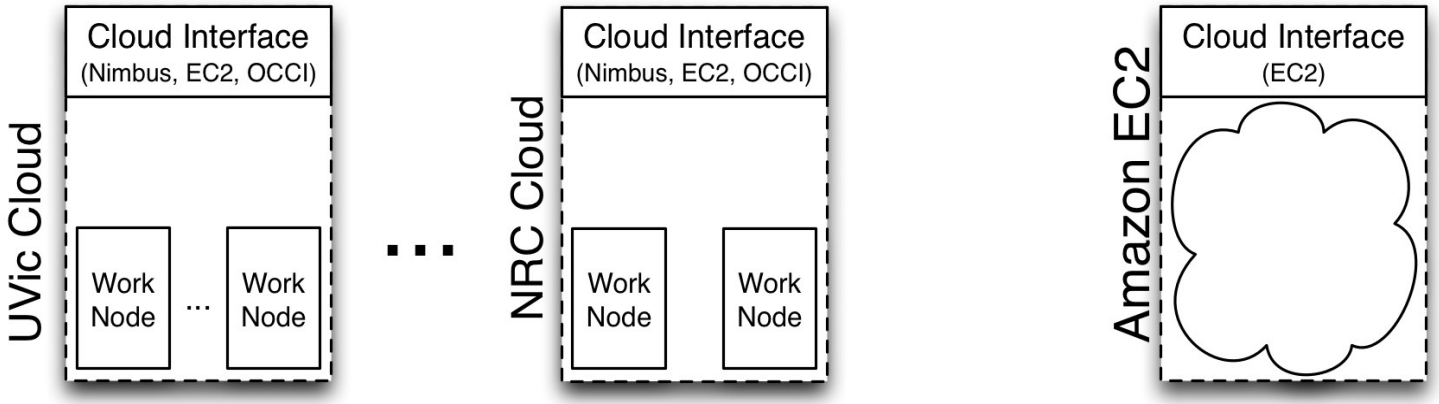
```
Executable = runpilot3-wrapper.sh
Arguments  = -s IAAS -h IAAS-cloudscheduler -p 25443 -w
https://pandaserver.cern.ch -j false -k 0

# Run-environment requirements
Requirements = VMType =?= "pandacernvm" && Target.Arch == "X86_64"

# User requirements
+VMName = "PandaCern"
+VMLoc = "http://images.heprc.uvic.ca/images/cernvm-batch-node-2.5.1-3-1-
x86_64.ext3.gz"
+VMMem = "18000"   #MB
+VMCPUCores = "8"
+VMStorage = "160"   #GB
+TargetClouds = "FGHotel,Hermes"
x509userproxy  = /tmp/atprd.proxy
```

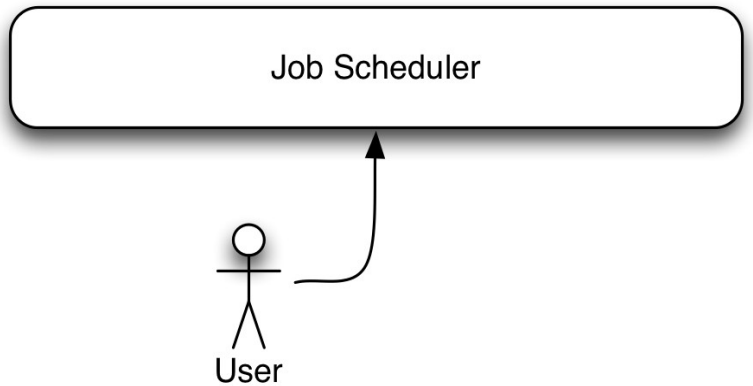
Step 1



Research and Commercial clouds made available through a cloud interface.

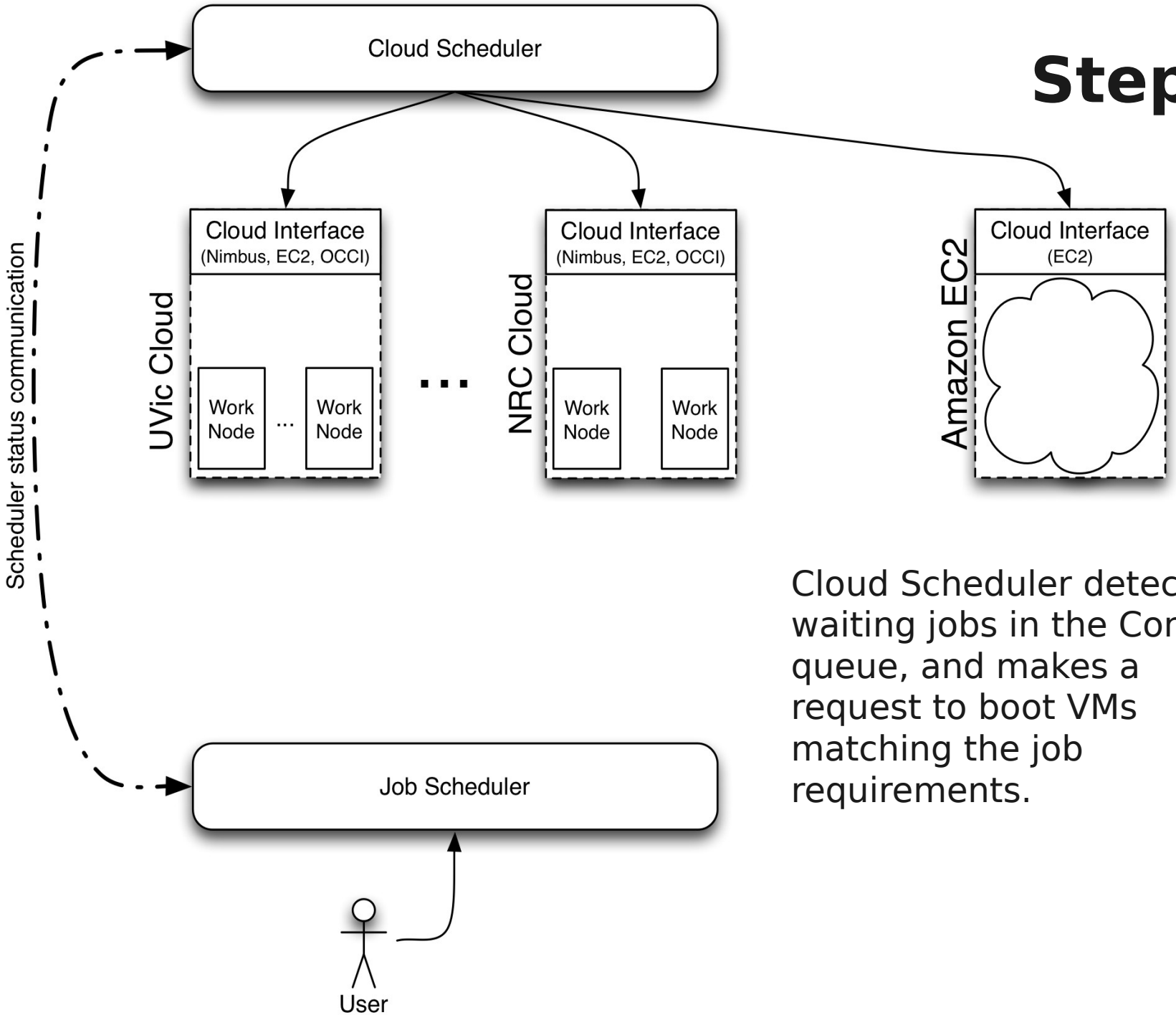
Step 2





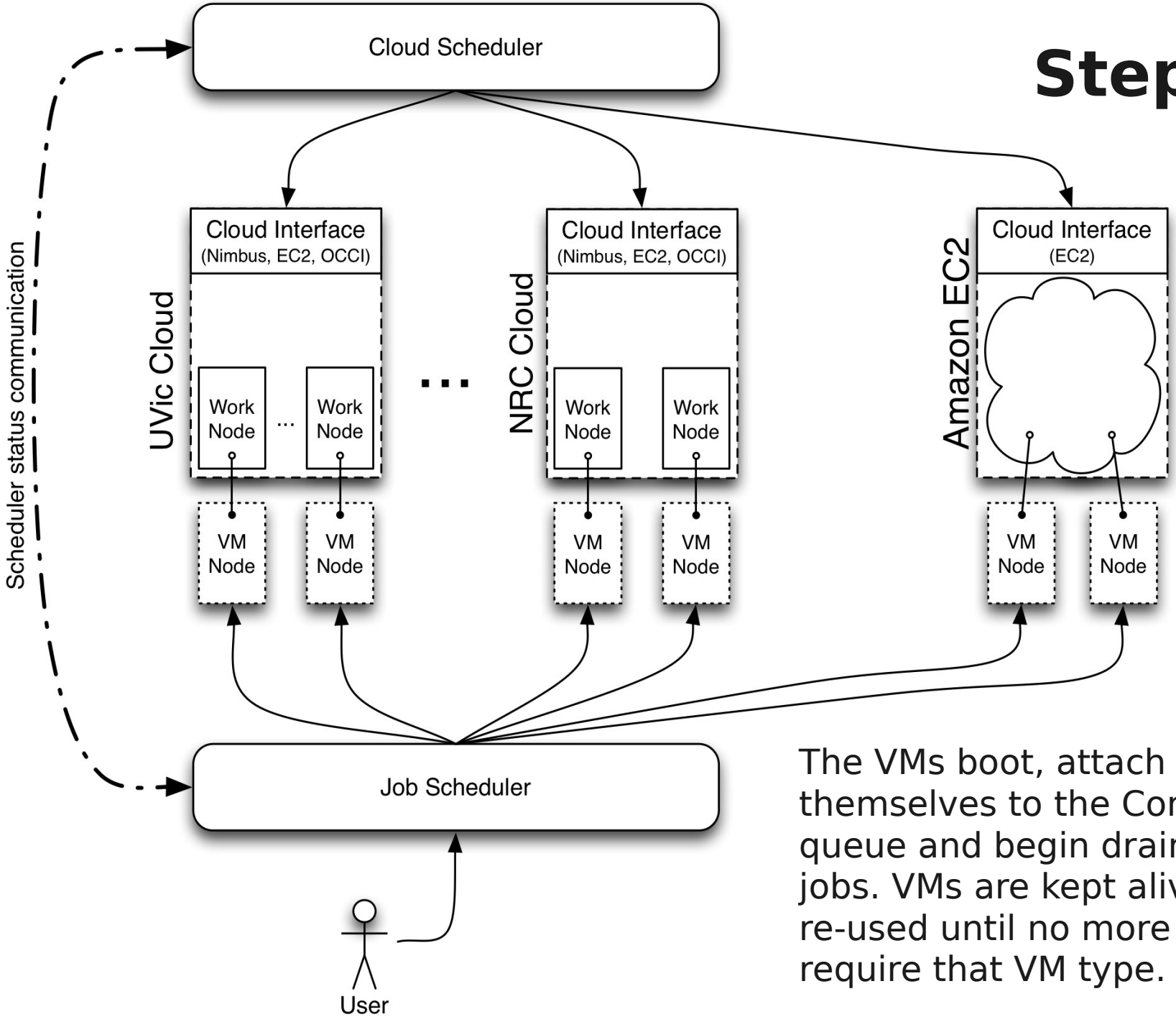
User submits a Condor job. The scheduler might not have any resources available to it yet.

Step 3



Cloud Scheduler detects waiting jobs in the Condor queue, and makes a request to boot VMs matching the job requirements.

Step 4



The VMs boot, attach themselves to the Condor queue and begin draining jobs. VMs are kept alive and re-used until no more jobs require that VM type.

Implementation Details

- Condor Job Scheduler
  - VMs contextualized with Condor Pool URL and service certificate
  - VM image has the Condor startd daemon installed, which advertises to the central manager at start
  - GSI host authentication used when VMs join pools
  - User credentials delegated to VMs after boot by job submission
  - Condor Connection Broker handles private IP clouds
- Cloud Scheduler
  - User proxy certs used for authenticating with IaaS service where possible (Nimbus). Otherwise using secret API key (EC2 Style).
  - Can communicate with Condor using SOAP interface (slow at scale) or via condor_q

Credential Transport