CHEP 2013

Cloud Bursting with glideinWMS

Means to satisfy ever increasing computing needs for Scientific Workflows

by
I. Sfiligoi[1], P. Mhashilkar[2], A. Tiradani[2],
B. Holzman[2], K. Larson[2] and M. Rynge[3]
[1]University of California San Diego [2]Fermi National Accelerator Laboratory
[3]University of Southern California, ISI

Some history

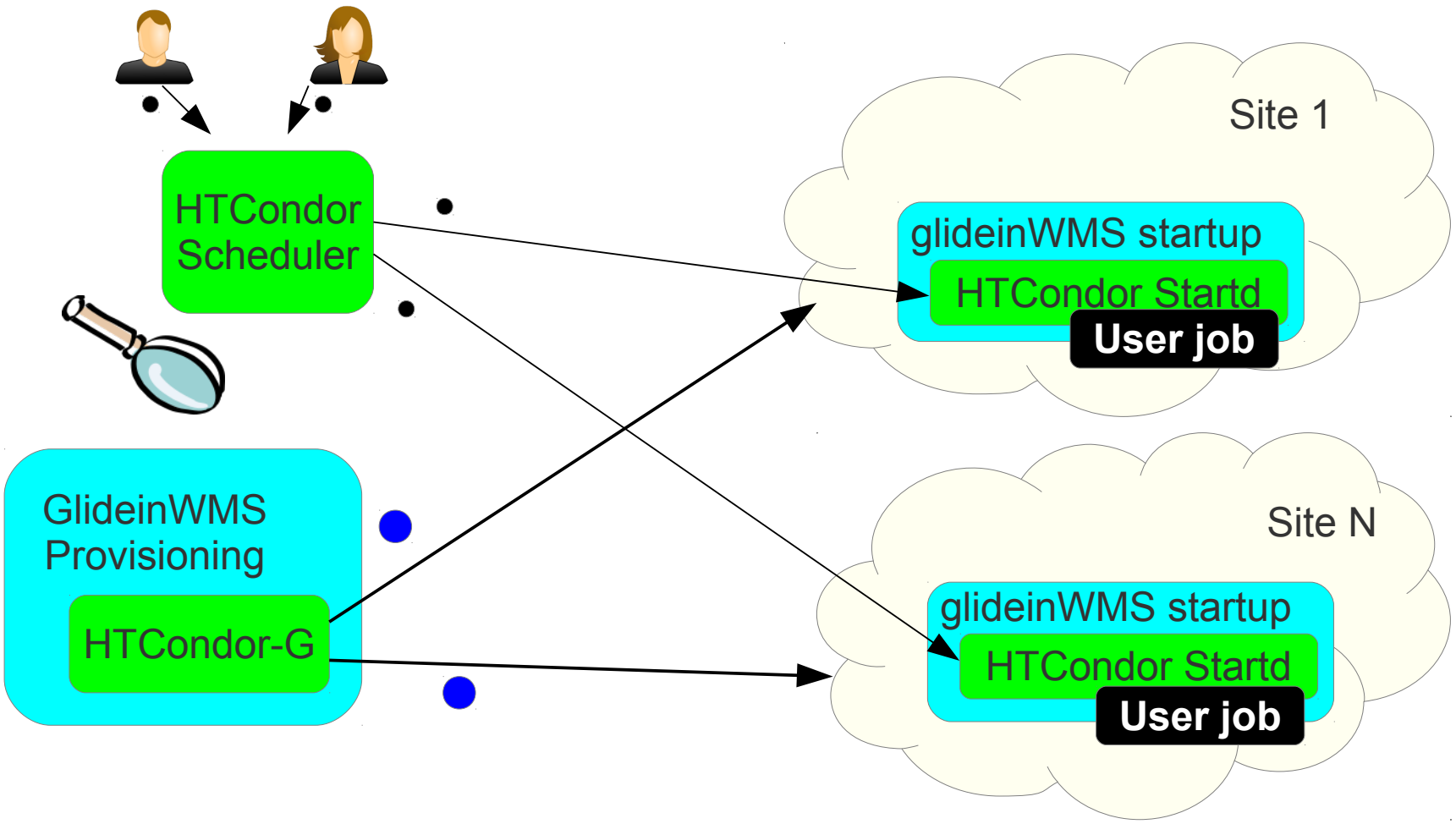
- Most major scientific communities have **outgrown single machines** a long ago
  - Distributed computing has become a must
- **Local clusters** started to pop up at institutes
  - But many communities fast outgrew the resources available on a single site
- Using **multiple clusters** became a need
  - “The Grid” created to provide a federated model
  - But job partitioning became a major problem

glideinWMS – A Pilot system

- **Users want a single cluster to submit to**
  - So let's create (a logical) one
- **The Pilot paradigm was born**
  - **Separates provisioning from scheduling**
  - Provisioning ~= Get ownership of a resource
  - Scheduling ~= Schedule a user job on that resource
- glideinWMS is a Pilot implementation
  - Build on top of HTCondor (formally known as Condor)

glideinWMS architecture

Very simplified version

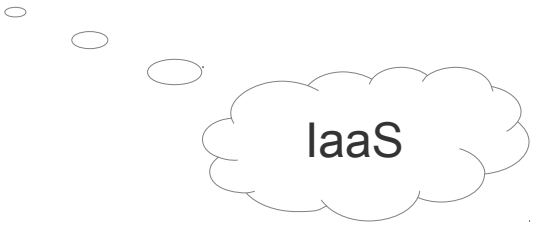


glideinWMS today

- glideinWMS is today the leading Pilot implementation in the Open Science Grid
  - More than ten VOs use it
- CMS uses it to submit both to OSG and EGI
  - Primary scheduling system for the past year

Moving beyond the Grid

- Cloud computing has emerged as a major new source of compute resources
  - Pioneered by Amazon with EC2
  - But many alternatives exist today



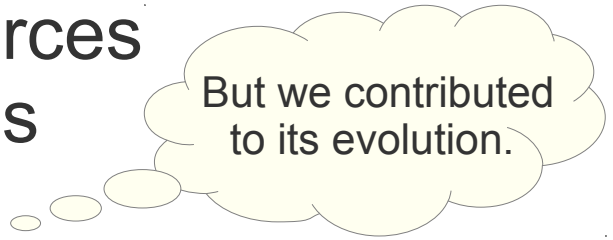
- **Cloud computing is conceptually similar to Grid computing**
  - But expects a full OS image, not just the application
- **Pilot infrastructures again essential**
  - Scientists just want to run jobs

How is Cloud different?

- There is the issue/opportunity of the OS image
- But the bigger issue is that virtually no Cloud provider offers Grid-compatible interfaces
  - Federated x509 credentials not accepted (with few exceptions)
  - EC2-compatible API instead of GRAM/CREAM
- Current state-of-the-art not great
  - Only partial API compatibility between implementations
  - No concept of credential federation

glideinWMS and Cloud provisioning

- glideinWMS always relied on **HTCondor-G for provisioning**
  - All Grid submissions already go through HTCondor-G
  - Adding logic to request Cloud resources was thus a minor code change for us
    - HTCondor-G does the heavy lifting



- **Configuring the resource** once we get it is instead something **we do**
  - Significant effort needed here

Configuring Cloud resources

- In the Grid, the WN dynamically gets at least
  - Executable
  - Arguments
  - x.509 proxy
- In the Cloud, the only dynamic part is the
  - UserData string
- glideinWMS had to **encode** Args+Proxy → UserData
  - We don't strictly need a dynamic executable

Cloud startup script

- As mentioned before, in the Grid one dynamically delivers the startup script
- In the Cloud, it is baked into the OS image

- We implemented it as **one of the services**
  - So it starts up during OS boot
- To keep uniformity, it is just a lightweight wrapper that downloads the real startup script from the glideinWMS instance and runs it

Missing functionality

- **In the Grid, it is normal to expect stdout and stderr of a job to be returned to the submitter**
  - glideinWMS was thus heavily relying on it for auditing purposes
- There is **nothing equivalent in the Cloud**
  - We still need to solve this part

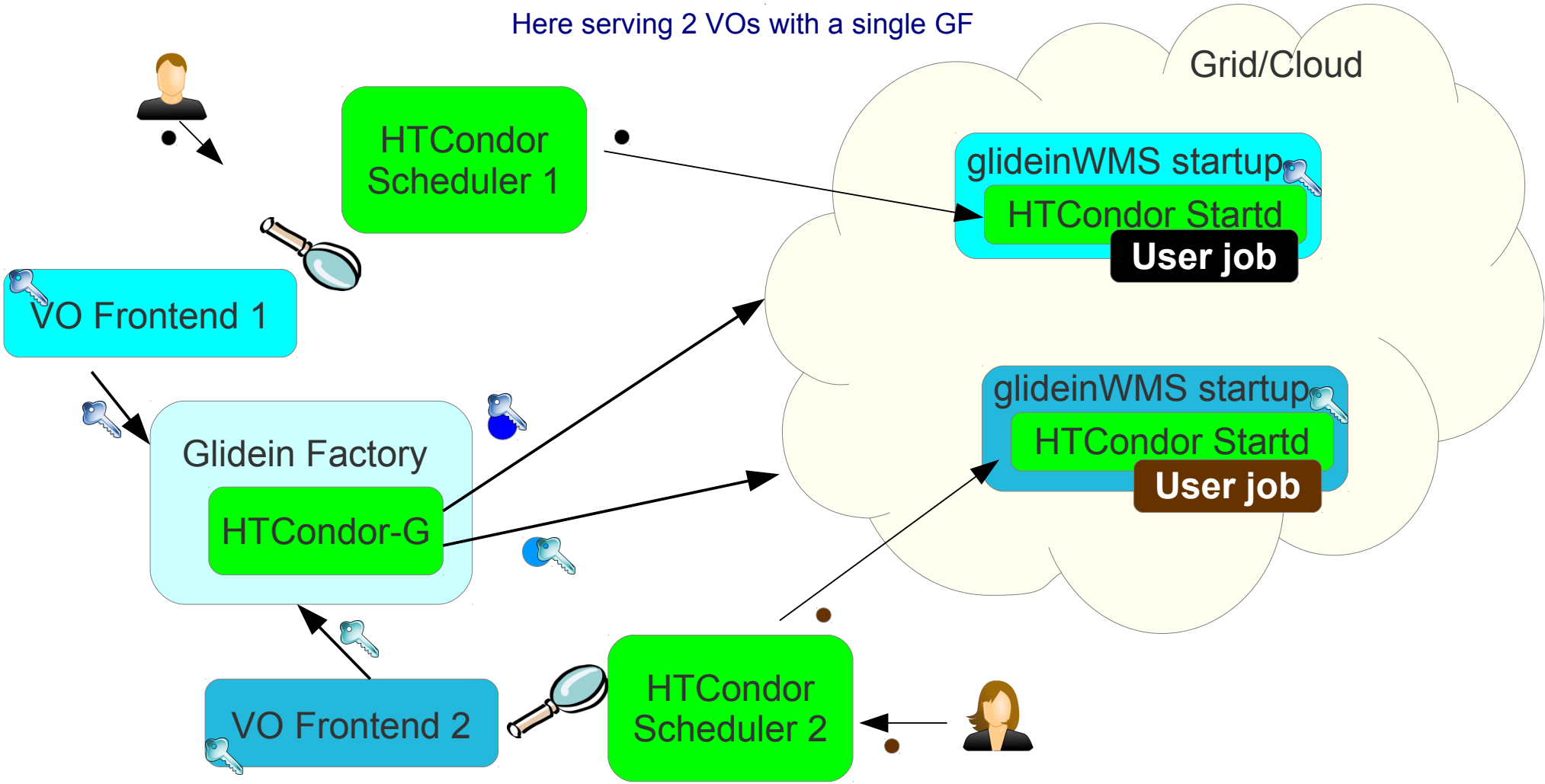
Internal changes

- The glideinWMS internal architecture calls for two distinct players
  - A glidein factory – Talks to the resource providers
  - A VO Frontend – Implements the provisioning logic
  - In N-to-M relationship
- The internal protocol was assuming Grid-type resources
- Had to **extend it** to support
  - Multiple credential types (i.e. not just x509)
  - Multiple trust domains (see next slide)
  - Optionally, VO-provided OS image

glideinWMS internals

in a very simplified picture

Here serving 2 VOs with a single GF



For more details, see: http://www.slideshare.net/igor_sfiligoi/glideinwms-training-jan-2012-glideinwms-architecture

Implications of multiple credentials

- **In the Grid, one proxy can be used to access any Grid site** (with very few exceptions)
- When you have a mix of Grid and Cloud resources, you will almost certainly **need multiple credentials** as well
  - i.e. an Amazon credential will not work at CERN
- glideinWMS solved the problem by introducing **trust domains**
  - A credential belonging to a trust domain is expected to be usable on all “sites” belonging to it
  - The provisioning logic will thus match on it

Presented functional prototype at CHEP2010

- The basic Cloud functionality was available in glideinWMS already during CHEP2010
  http://iopscience.iop.org/1742-6596/331/6/062014
- But the devil is in the details!
- And most of those details are not even under our control
  - Basically, various Cloud Middlewares are not fully implementing the “Cloud specs”

Required close collaboration with HTCondor team

Cloud Middlewares

- Amazon EC2 is of course the most famous one
  - If that was the only Cloud we needed to support, the CHEP'10 code was *almost* ready for prime-time
- But most scientific communities seemed more interested in other Middlewares
  - ANL's Magellan based on Eucalyptus
  - CERN's HLT based on OpenStack
  - Fermilab's FermiCloud based on OpenNebula

Issues along the road

- Three categories
  - EC2 Submission API issues
  - EC2 Runtime issues
  - Scalability issues

The EC2 submission API

- The non-Amazon Cloud Middlewares have a very loose interpretation of the EC2 API semantics
  - 2010 HTCondor-G would simply not work
  - Required extended collaboration with HTCondor team
    - But now OpenStack and OpenNebula usable
- A couple concrete examples:
  - API calls not idempotent
  - VMs refuse to properly terminate

The EC2 Runtime environment

- Each Cloud Middleware provides different ways to contextualize the OS image
  - Not even a common API
  - Each time we add a new Cloud provider, we have to discover how to use it
  - Our startup script has to have different execution paths for different Middlewares
- Concrete example:
  - There is no uniform way to get the UserData into the Cloud instance

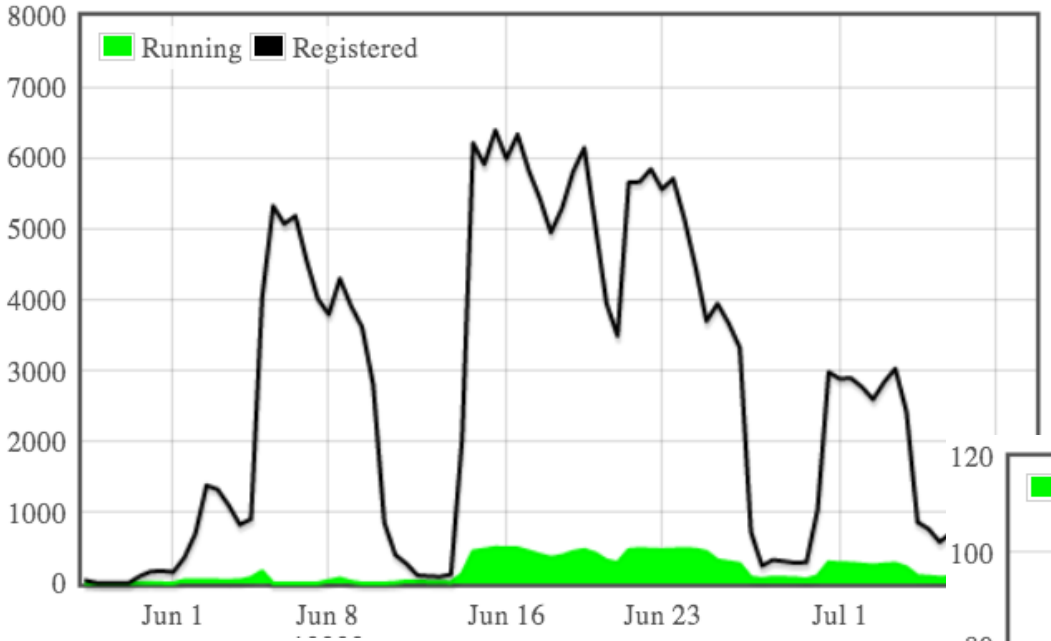
Scalability issues

- Every time we tried to get a significant amount of resources out of Cloud providers, we ended killing the service
  - Again, close collaboration with HTCondor team to mitigate the problem until bearable
- Concrete example:
  - OpenStack's Nova scheduler seems to be limited to 500 polling requests every 5 minutes

Deployment plans

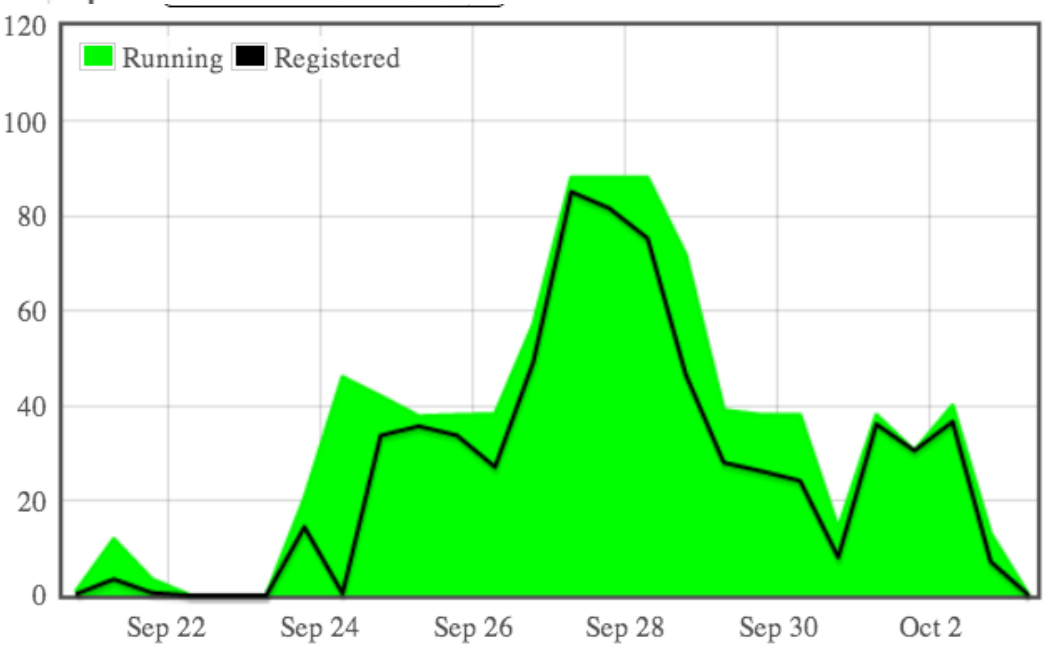
- CMS has been running an advanced beta of the Cloud-enabled glideinWMS for about a year on the CERN's HLT farm
- NOVA has been test-using it on FermiCloud since early Sep'13
- The Cloud-enabled glideinWMS was declared production quality early Oct'13
  - And has been put in production soon after on a OSG glidein factory

A few graphs



← CMS on CERN's HLT over Openstack
up to 500 VMs
up to 6.1k cores

NOVA on FermiCloud over OpenNebula →
up to 90 VMs
up to 90 cores



Summary

- The Cloud is conceptually similar to the Grid so creating a Pilot-based overlay makes sense
  - But different enough to require significant internal changes in glideinWMS
- Moving between Cloud providers hard due to significant implementation differences
  - Required significant workarounds to be usable
- glideinWMS has helped CMS using Cloud resources for about a year
  - And now available for other VOs as well

Acknowledgements

- Fermilab is operated by Fermi Research Alliance, LLC under Contract number DE-AC02-07CH11359 with the United States Department of Energy (DOE).
- The work was partially sponsored by
  - DOE and KISTI under a joint Cooperative Research and Development Agreement CRADA-FRA 2013-0001 / KISTI-C13013
  - US National Science Foundation (NSF) grants PHY-1120138 and OCI-0943725.