Introduction to glideinWMS

From the point of view of the CMS VO

by Igor Sfiligoi (UCSD)
HTC, DHTC and overlays

- **High Throughput Computing** is about efficient and effective scheduling of user jobs on top of a compute resource pool.

- **Distributed HTC** is about aggregation of many unrelated HTC systems.
  - With overlays being the preferred operation mode.
Overlays in DHTC

- A DHTC overlay is about creating a global HTC system on top of leased resources.
HTC vs DHTC products

• There are several mainstream HTC products available, e.g.
  • Condor
  • PBS, with variants like Torque/Maui
  • LSF
  • SGE, also known as Oracle Grid Engine
• None natively supports DHTC overlays
  • They all assume full control of the managed compute resources
DHTC overlay systems

- There are a few full-stack DHTC overlay systems available, e.g.
  - DIRAC
  - PANDA
  - Alien
- glideinWMS instead heavily relies on HTCondor for the HTC part
  - glideinWMS proper only handles the D- part

i.e. they implement all the bits and pieces needed to make the DHTC overlay logic work
Pros and cons of layered approach

- The glideinWMS approach of relying on HTCondor for the HTC part has its benefits
  - Very mature HTC layer, with “standard” tools
  - Relatively low development and maintenance cost for the glideinWMS-proper code
- But it does have its disadvantages
  - Less integration between the two layers
  - The need to work around some “dedicated HTC” assumptions in HTCondor
Resource provisioning

- The glideinWMS-proper part is mostly about **resource provisioning**
  - Deciding *when* more resources are needed,
  - Deciding *where* to lease the from
  - Validating and configuring them after the lease
- It also assists in deciding when resources should be returned to the resource owner
  - Although HTCondor itself implements the core logic
Resource provisioning

Resource provisioning

glideinWMS

HTCondor

HTC
Grid and Cloud

- Grids and Clouds are the two main paradigms for providing non-dedicated resources
  - Both offer resource elasticity

- Grid computing is basically a federation of HTC clusters
  - Thus a true Distributed HTC

- Job queuing is a native paradigm

- (Commercial) Clouds are about leasing resources on a pay-as-you-go basis
  - And they happen to use virtualization

  - Instances expected to start almost immediately
Grid and Cloud

- Grids and Clouds are the two main paradigms for providing non-dedicated resources
- Job queuing is a native paradigm
- Job queuing is a native paradigm
- Thus a true Distributed HTC
- And they happen to use virtualization
- Instances expected to start almost immediately

Note:
So-called “scientific clouds” are typically just Grid systems that use virtualization (and a different middleware stack)

Notes:

- glideinWMS is currently optimized for the Grid model
- (Commercial) Clouds are about leasing resources on a pay-as-you-go basis
- And they happen to use virtualization
R.P. as pilot job submission

HTCondor-based pilot jobs = glideins

glideinWMS

Central manager

Condor

Submit node
Condor
Submit node
Submit node

Grid

Execute node
Execute node
Execute node
Execute node
Execute node

Condor
Two layers

- glideinWMS is composed of two layers
  - **Glidein Factory** – The abstraction layer
  - **VO Fronend** – The policy layer - i.e. the brain

- The splitting in two allows for the **Glidein Factory** to be *generic*
  - It can thus serve many different VO Frontends
  - It can (and should) be shared between many independent VOs

- A VO Frontend can also use more than one Glidein Factory
  - For scalability and reliability
The VO Frontend

• The name may be misleading
  • It is really the “matchmaker of Grid resources”

• Introduces a new quanta
  • **Entry** – logical equivalent of a “queue at a site”
    Basic working block of a G.F.

• The VO Frontend
  1) **Matches idle Jobs to Entries**
  2) **Instructs the affected G.F.** to increase or decrease
    the number of glideins on that Entry

Thus regulates the resource provisioning
Glidein Factory

- The G.F. is really just an **abstraction layer**
  - Provides a logical description of the resources
    - As a set of attributes
  - Insulates the Frontend from the resource details
    - e.g. knowing the nodes a site uses for job submission
  - Allows new technology to be added seamlessly (e.g. GRAM vs CREAM and interfacing to Clouds)
- It also provides a **troubleshooting service**
  - The factory operators are supposed to **address any Grid related problems they observe**
    - e.g. missing libraries or firewall problems
Working relationship

- The Glidein Factory is just a slave
  - It will do whatever the VO Frontend asks it to do
  - e.g. no G.F. submission logic or throttling
- The glideins are submitted in the VO Frontend's name
  - The VO Frontend owns the credential (e.g. x509 proxy)
  - And delegates it so the G.F. can submit glideins
Typical VO Frontend setup

The CMS VO currently uses 4 G.F.s, 3 of which are shared among many VOs.
Two level matchmaking

- In a glideinWMS system there are **two layers** that do job matchmaking
  - The VO Frontend, for resource provisioning
  - The HTCondor Central Manager for HTC
- The two layers **must be in sync**, else
  - Jobs may never start, since no resources are provisioned, or
  - Provisioned resources are never used, since no jobs match
Matchmaking logic defined in VO FE

• The VO Frontend contains both
  • Its own matchmaking logic vs Entries, and
  • the HTCondor matchmaking logic, in the form of Machine Requirements
• The two may be slightly different, since Entries have less detail than Machines

• User jobs should not define any Requirements
  • Since those are only used by HTCondor
  • They should only define attributes describing the desired outcome
**Limited lease lifetime**

- glideinWMS operates on leased resources
  - And all *leases have a limited lifetime*
- In the Grid, currently, the leases are quite short
  - Typically 24h to 48h
  - And cannot be extended at runtime
- All user jobs must terminate before the lease end
  - Or they will killed
- Each glidein can run multiple jobs
  - *Policy must prevent matching jobs that cannot finish in time*
Multi-core glideins

- A single glidein can lease multiple CPUs
  - And then partition them among several jobs
- Be wary of the **termination waste**, though
  - Since unlikely that all jobs terminate together
Security considerations

- Pilot jobs typically don't have superuser privileges
  - At least in most of the Grid ecosystem
- HTC system needs system-level protections
  - To protect itself from the user jobs
  - To protect jobs from different users, running on the same node
    - Most nodes are multi-core these days
- Most of CMS sites support gLExec
  - Allows UID switching by presenting a user x509 proxy
  - Requires user jobs to have a valid proxy at all times
Security considerations

- DHTC has nodes that communicate via WAN
  - Which cannot be considered safe on face value
- All glideinWMS processes, including HTCondor, enable for all network communication
  - Strong authentication methods (based on x509 proxies)
  - Integrity checks
- Network security can be expensive
  - The setup has to account for it
As a reminder, glideinWMS creates an overlay DHTC system on top of resources provided by many sites.
The big picture

- The previous slide could lead you to believe that there can only be one overlay.
- In reality, **there can be any number of them!**
  - Each serving its own user community.
  - Can be other DHTC technologies, too.
- Not necessarily all using the same set of Sites, i.e. HTC clusters.
Why many DHTC instances?

• Different VOs typically don't want to share a glideinWMS DHTC instance

• While technically possible, there are
  • Security risks
  • Increased maintenance complexity
  • Politics!

• Typically we get one glideinWMS DHTC instance per VO
  • But some VOs have more than one

Immagine co-scheduling of CMS and ATLAS jobs!

The CMS VO currently has 3
Pointers

- glideinWMS development team is reachable at glideinwms-support@fnal.gov
- The official project Web page is http://tinyurl.com/glideinWMS
- HTCondor Web page http://research.cs.wisc.edu/htcondor/
Acknowledgments

- This document was sponsored by grants from the US NSF and US DOE, and by the UC system