Dynamic allocation and Cloud Computing in CMS

Massimo Sgaravatto – INFN Padova

Claudio Grandi – INFN Bologna

CMS VIEW ON CLOUDS

- From January GDB meeting
  - CMS is reasonably happy with the current resource allocation system in use on CMS-owned resources
  - Including shared sites where CMS owns a fraction of the resources
  - Any change, including the use of a Cloud interface, is acceptable provided there is no significant efficiency degradation
  - Commercial or in general opportunistic Clouds may be interesting for absorbing usage peaks
  - CMS is active in adapting the job submission framework to Cloud interfaces

CMS VIEW ON CLOUDS (CONT.ED)

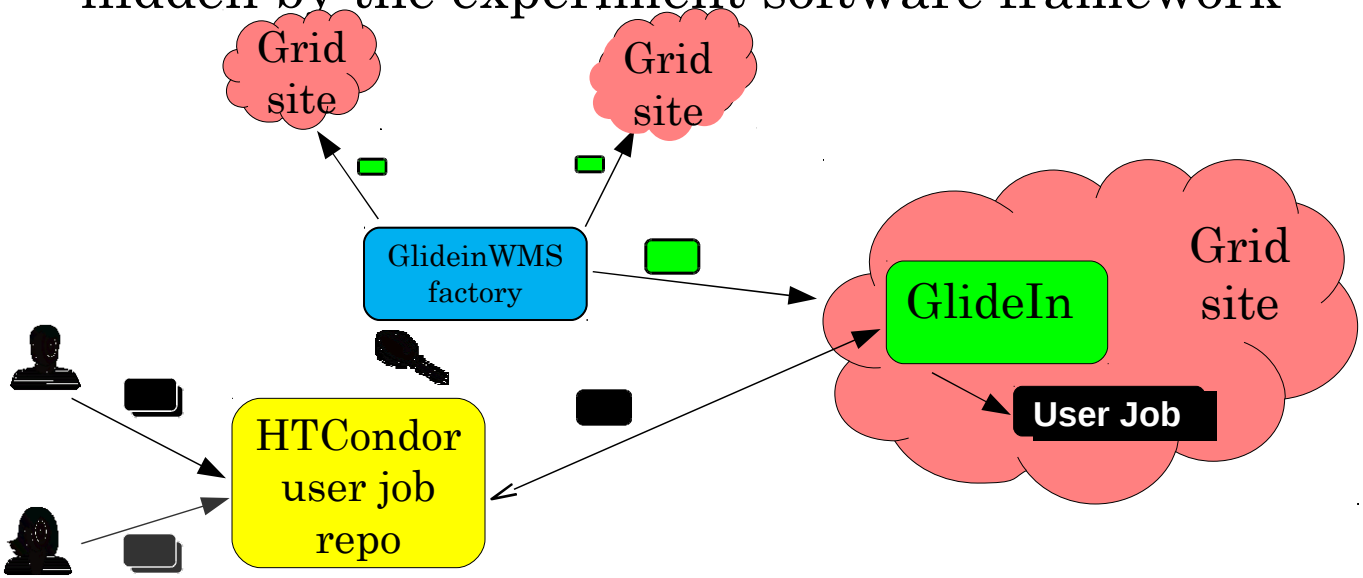
- From Spring offline and computing week
  - We should not go to heavy in-house engineering in order to solve problems already solved by current (old?) technologies. Either our requirements are addressed by the "cloud community" or we try to exploit what we have. In the end going to cloud for us is a way to use commonly used software and thus reduce the need to develop specific solutions.

CMS Job Submission Framework

- Mainly based on the pilot job approach
  - Separate resource provisioning from resource scheduling
  - Pilot jobs (not user jobs) are submitted as normal Grid jobs to the available sites to create an overlay batch system
  - User jobs are then run on this overlay batch system, whose size changes according to the resource availability in the Grid
- Implemented via GlideinWMS
  - Heavily HTCondor based
    - The overlay batch system is HTCondor
  - The pilot, called glidein, basically installs and configure the batch slot as a Condor executing machine
  - Implemented by USCMS. Support currently provided by FNAL

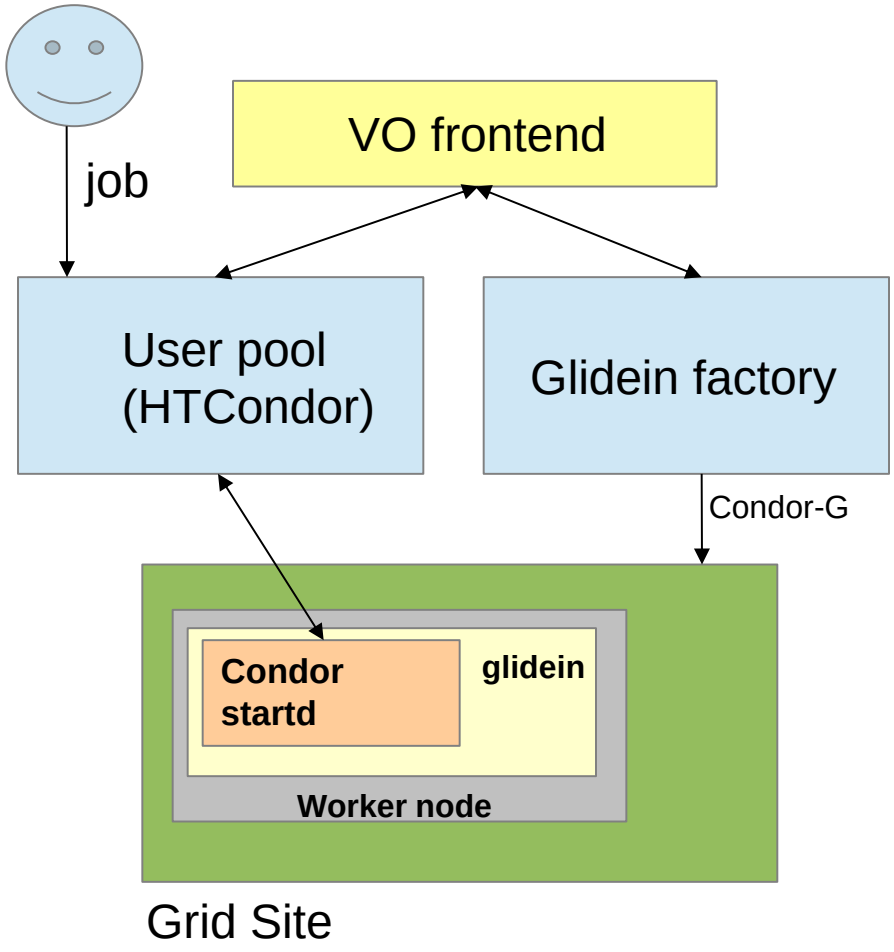
GLIDEINWMS OVERVIEW

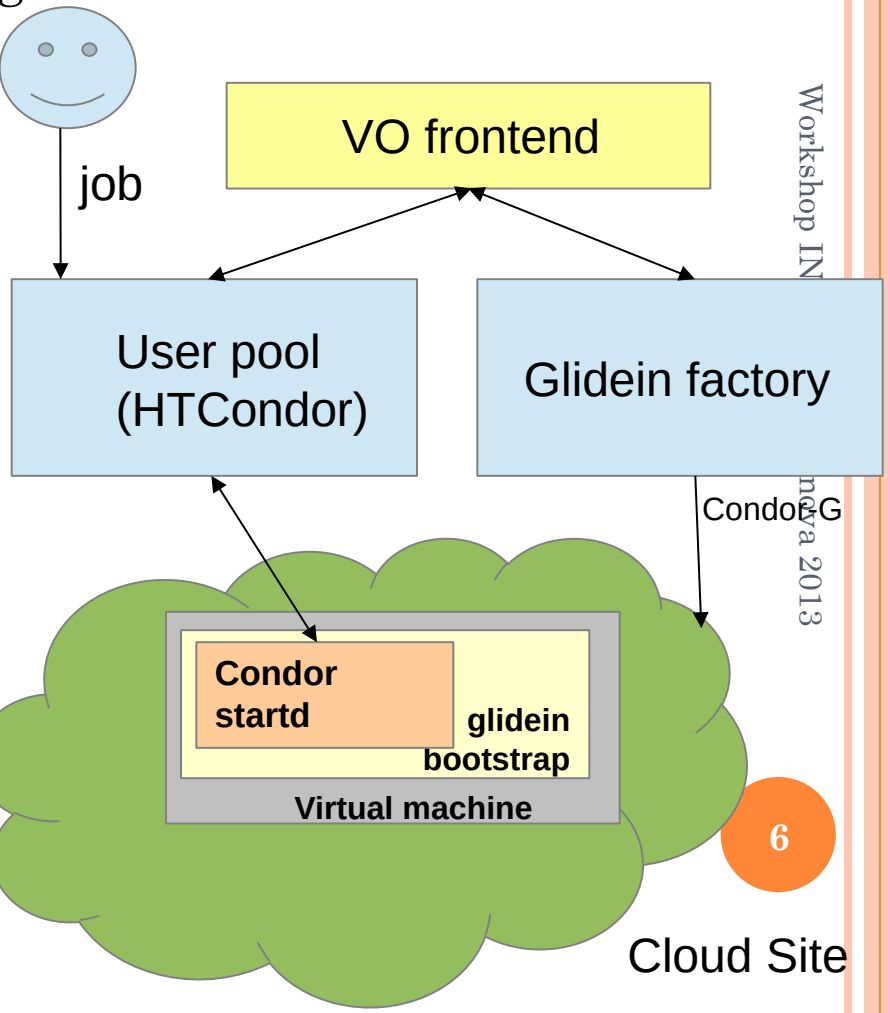
- Pilots are submitted to all Grid sites by GlideinWMS factories
- As soon as a pilot (glidein) starts running, that allocated job slot joins the HTCondor user pool
- User jobs are Condor jobs, submitted to HTCondor user pool
  - Even if users don't realize to use HTCondor, since this is hidden by the experiment software framework



Dynamic provisioning on Grid & Cloud

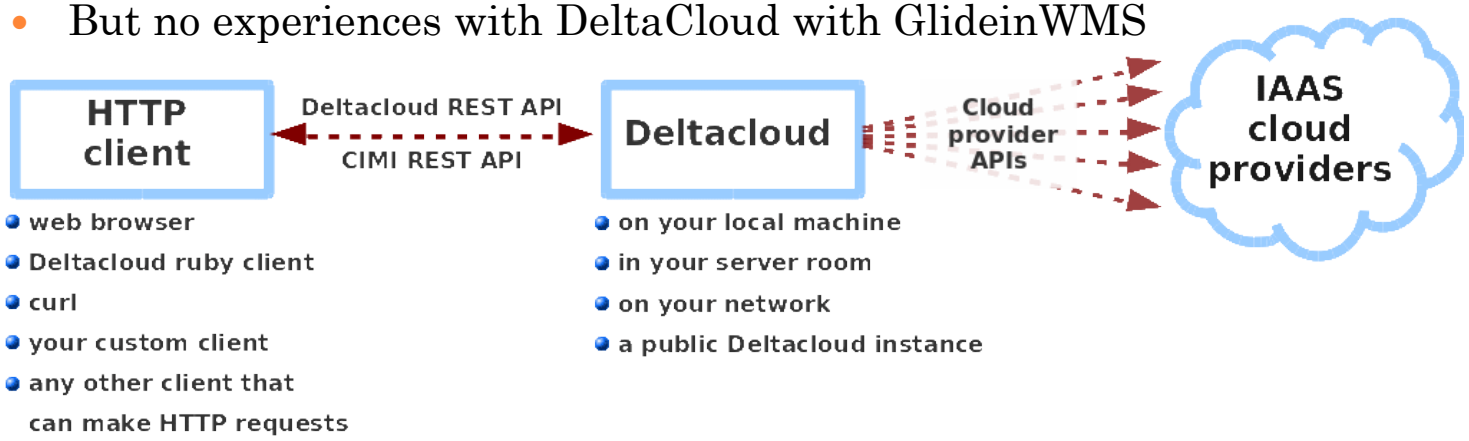
- Cloud support introduced with GlideinWMS v3
  - Same architecture: submission of Glideins vs instantation of VMs where glideins run





GLIDEINWMS &CLOUDS

- HTCondor (and GlideinWMS) support Clouds providing an EC2 interface
  - This does not mean that GlideinWMS works out of the box with any EC2 based cloud
    - Tested with Amazon EC2, Openstack, Eucalyptus
- Actually HTCondor supports also DeltaCloud (Cloud "translation" service)
  - But no experiences with DeltaCloud with GlideinWMS



- AFAIK no plans to support OCCI in HTCondor (and therefore in GlideinWMS)

IMAGES

- VMs instantiated by GlideinWMS factory
- Images must be registered in advance in the relevant Clouds
- Glidein bootstrap RPMs must be part of the VM image
- Image id, flavor, and EC2 credentials specified in the GlideinWMS Frontend or Factory configuration file
- GlideinWMS creates a different SSH key for each requested VM
  - If a key is compromised, only a VM is affected
  - But this also means that troubleshooting problems in the VM is not straightforward, unless you have access to the GlideinWMS factory

CONTEXTUALIZATION

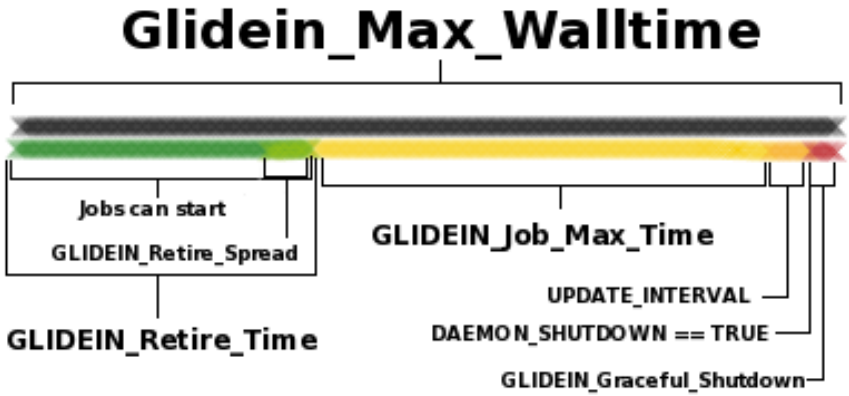
- User-data used only for Glidein configuration
  - Tarball with glidein startup script, proxy, details about factory and frontend
- Normal site contextualization is not possible
  - Difficult to have a single "certified" image usable everywhere + specific local customization
  - However some specific site (entry) customizations scripts can be defined at the GlideinWMS level
  - Issue planned to be addressed in GlideinWMS after V3.1

GLIDEIN IN CLOUD: HOW IT WORKS

- Glidein started as service when the VM is created
- Everything runs as an unprivileges user (created by glidein RPM prescript)
- Glidein configuration in user-data
- Download startup script
- Perform checks defined in the Factory and/or Frontend glideinWMS
- Download and execute HTCondor startd
- Startd runs multiple single core jobs and/or multi-core jobs to “fill” the machine
- Glidein stops its execution in case of error or when there is no more work
  - VM stopped and deleted

VM Lifetime

- Glidein lifetime is VM lifetime
- Configurable at GlideinWMS level in the very same way as 'normal' glideins
- Possibility to configure
  - Maxixum time the VM can be idle before shutting down
  - Retire time: no new user jobs are accepted after this time
  - Job Max Time: after Retire time is exceeded, user jobs can run at most for this time otherwise they are killed
  - Max wall time: maximum overall time for the VM



TESTING AT AI@CERN

- AgileInfrastructure@CERN
  - OpenStack based IaaS cloud
  - CERNVM based images
  - CVMFS for experiment software
  - Xrootd for I/O from/to EOS
- First test with VM instantiation and job submission manually handled
  - MonteCarlo and analysis jobs
  - Very high job efficiency

TESTING AT AI@CERN (CONT.ED)

- Second test using GlideinWMS
  - 800 cores
    - Started with 8 cores -16 GB images; then moved to 4 cores - 8 GB images for faster allocation time
  - Montecarlo jobs (WMAgent)
    - Workflows completed without major problems
  - Analysis jobs (CRAB2 + RemoteGlidein)
    - 9581 jobs, 96% efficiency
  - Issue with high load in Clod Controller
    - Addressed in GlideinWMS with bulk queries and decreasing polling rate

The CMS HLT getting CMSoooooCloud

- CMS High Level Trigger Farm
  - 13.3K cores, ~ 195 KHS06
  - Not shared with other experiments
- Turning this farm into a Cloud
  - To use it during LS1 and in general whenever not used
  - Overlay infrastructure:
    - Minimal changes to convert HLT nodes in Cloud compute nodes
- Cloud middleware
  - OpenStack Essex, moving to Grizzly
    - 1 Cloud controller
    - 1 Glance server (images)
    - 1300 Compute nodes
      - KVM virtualization
  - Openvswitch used to virtualize network
- CMS Openstack, OpenSwitch-ed, Opportunistic, Overlay, Online-cluster Cloud (CMSoooooCloud)

HLT-Cloud testing

- Images
  - Created initially using CERNVM, then moved to BoxGrinder
    - CERNVM uses Conary instead of RPMs
    - SL 5.x + CA certificates + CVMFS + Glidein bootstrap + xrootd client
  - CMS software through CVMFS
  - 16 cores VMs
- I/O from/to EOS using xrootd
- Glexec not used
  - But there are no technical problems to use it, provided that the VM is properly configure to use it
- VM instantiations and job submissions managed using GlideinWMS v3
- Jobs submitted using WMAgent

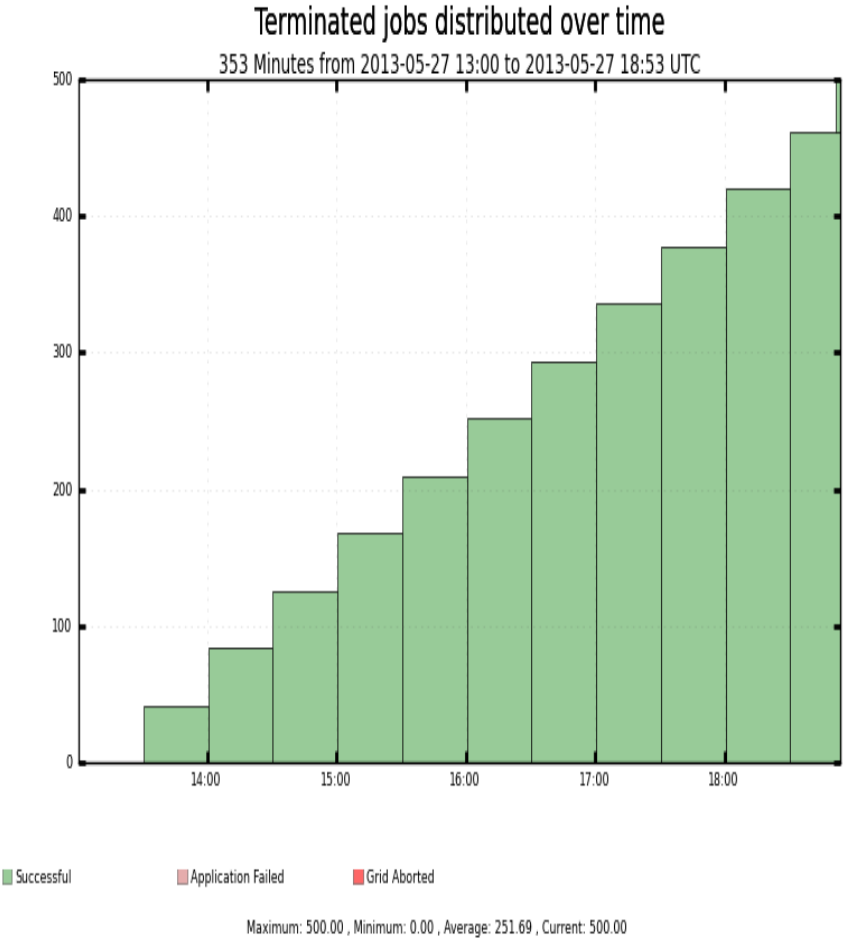
HLT Cloud testing (cont.ed)

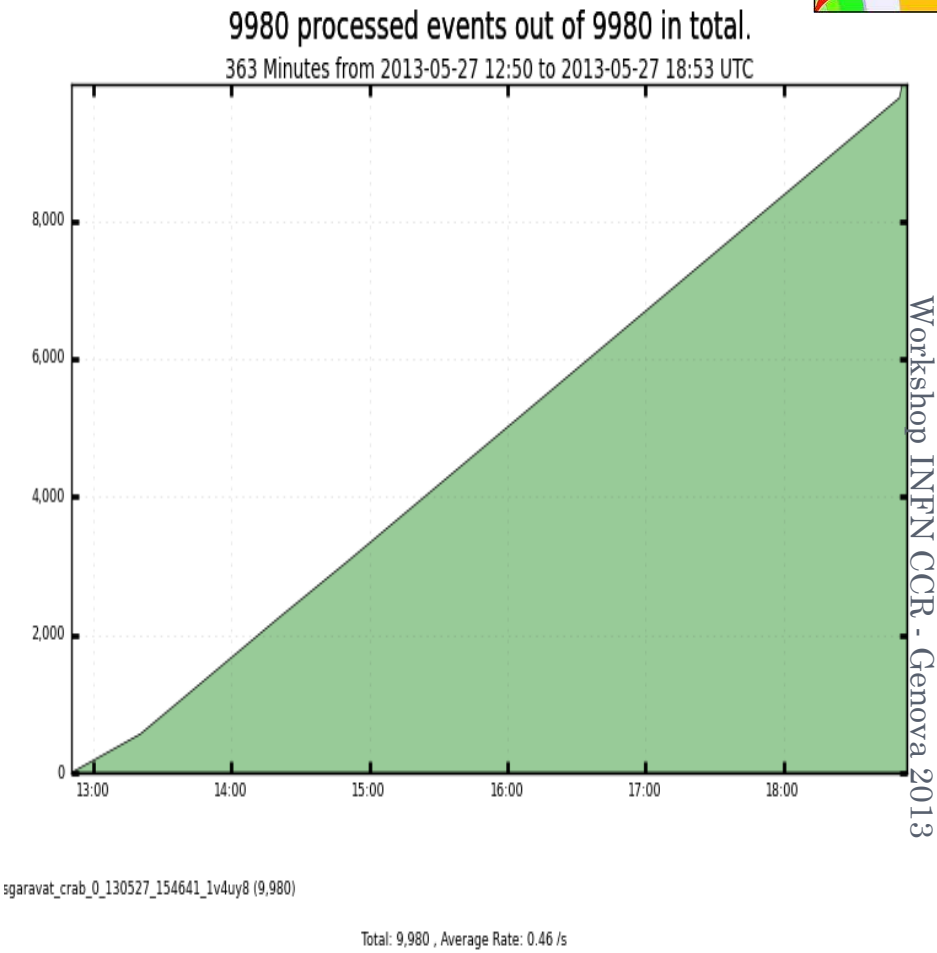
- Many issues, now basically addressed, e.g.:
  - CVMFS not available fast enough
  - Buggy xrootd client in the EPEL repo installed in the VM image
  - Termination of stopped VMs was not performed
    - Openstack ignoring the InstanceInstantiatedShutdownBehavior EC2 setting
  - Network saturation preventing to scale up
    - Network update in progress
- Reprocessing of 2011 data as validation task to start soon

Other CMS Cloud related activities

- GridPP Cloud Pilot in UK
  - Replicated HLT setup in smaller scale
    - GlideinWMS @ RAL
    - Openstack Cloud @ IC (7 compute nodes, 200 cores)
  - Being tested with CRAB2 analysis jobs
  - The relevant people are the same active in the HLT-cloud testing activities
- In Italy
  - Also replicating (in smaller scale) the CMS-HLT setup …
    - Glidein WMS instance at T2 Padova-Legnaro for Cloud testing
    - Small Folsom Openstack installation at T2 Padova-Legnaro
      - 1 controller node + 2 Compute nodes
    - Starting using also a small Grizzly Openstack installation at CNAF
    - Testing with CRAB2 & RemoteGlidein
  - … but the final goal is to evaluate something different (see later)

TESTING IN ITALY





CRAB2 jobs using OpenStack @ T2 Padova-Legnaro
Data read using xrootd, write using SRM

Can a Cloud replace a Grid CE ?

- E.g. for a T2 ?
- Some pros
  - Big (and increasing) Cloud community
    - In most of the cases we can hopefully see implemented (instead of implementing) what is needed
    - Less resources available for Grid middleware maintenance and evolution now
  - Easier to manage different requirements in terms of OS and libraries by the different VOs
  - Site admins relieved from lot of work
    - They have just to install and maintain an OpenStack (or whatever) installation
    - No middleware installation/update, no experiment software to install/configure, etc.
- Some cons
  - Site admins give up a degree of control on their resources
    - It is even not straightforward to access the VMs
  - **Fair-share management in sites supporting multiple VOs**

SCHEDULING

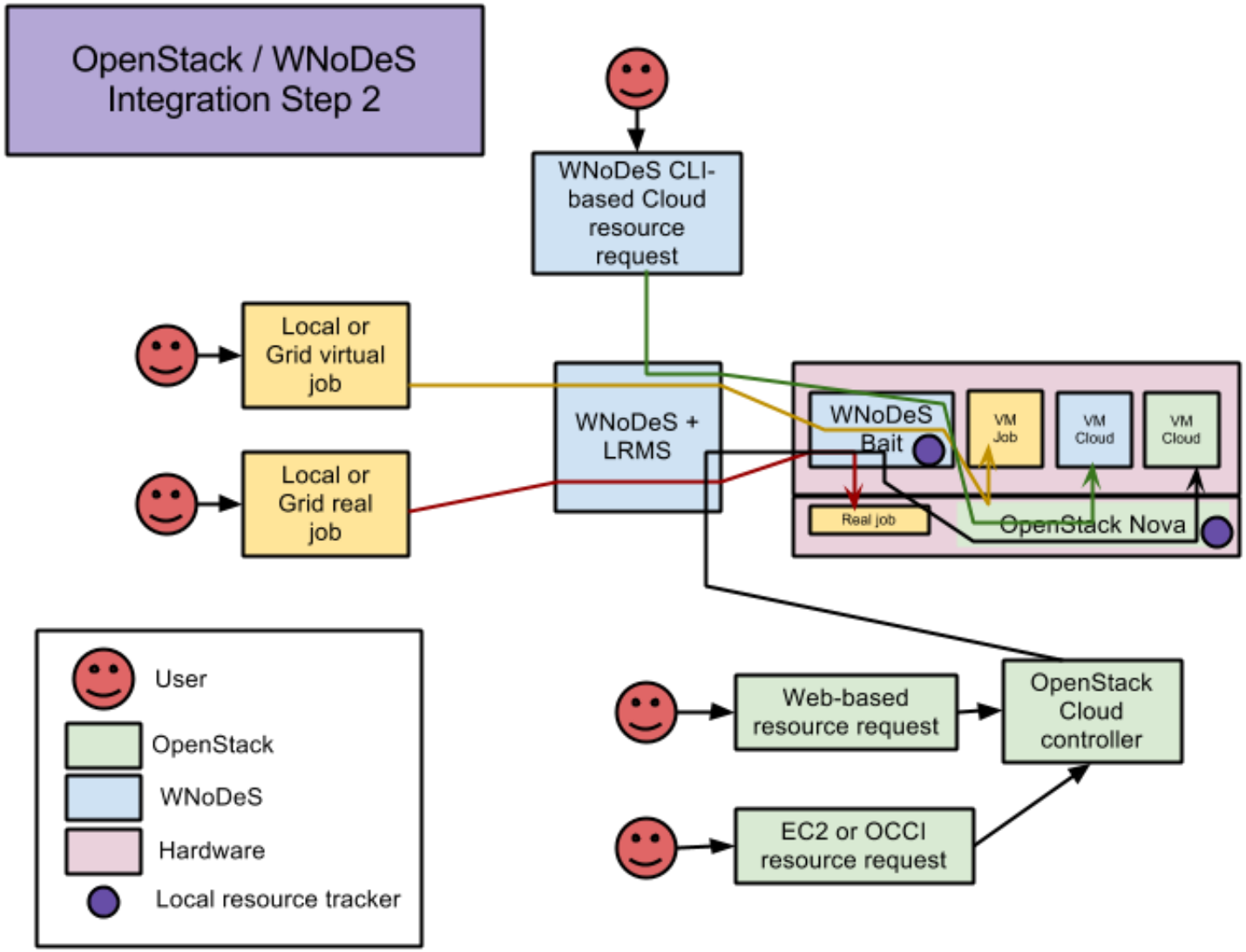
- In Grid fair-share among the VOs is implemented by the batch system
- This is something which doesn't exist in the IaaS clouds that were not designed as job submission facilities
  - There is not a queuing system for VM requests
  - If there are no more resources, the request to instantiate a new VM simply fails
- With the current implementations the only solution would be to statically partition the resources, which is something we don't want

SCHEDULING (CONT.ED)

- Some solutions proposed in the context of the WLCG GDB cloud working group, but they would require significant efforts for their implementations:
  - Implementation via graceful termination of VMs
    - VMs can be shutdown by the site if/when needed
    - SLA specifying X minimum hours of shutdown notice for VMs
    - VO A let the site knows they would like more resources
    - Graceful termination of VMs for VO B that were using unused resources
  - Economic model
    - VOs are given credits
    - The price of a VM increases with its duration an the number of VMs owned by a VO

WACK

- WACK supposed to address this issue
  - Wnodes + OpenStack integration
  - Exposing (also) EC2 interface therefore usable by GlideinWMS
  - Wnodes engine used to instantiate VMs
    - Using the batch system where fair-share can be implemented
  - See Davide's talk
- Agreed to try GlideinWMS → WACK interactions as soon as some WACK prototype appears


OpenStack / WNoDeS Integration Step 2
WNoDeS CLI-based Cloud resource request
Local or Grid virtual job
Local or Grid real job
WNoDeS + LRMS
WNoDeS Bait
VM Job
VM Cloud
VM Cloud
Real job
OpenStack Nova
User
OpenStack
WNoDeS
Hardware
Local resource tracker
Web-based resource request
EC2 or OCCI resource request
OpenStack Cloud controller

CONCLUSIONS

- Overall job submission architecture not changed to support Clouds
  - Dynamic provisioning of VM for Clouds
  - Dynamic provisioning of job slots for Grids
- Not far to be production ready for the use of opportunistic Clouds where scheduling is not needed
  - E.g. HLT farm
- Old technology (i.e. Grid) is probably still better for sites supporting multi-VOs
  - Not straitghforward to assure fair-share and efficient resource usage with the existing implementations

BACKUP SLIDES

GLIDEINWMS ARCHITECTURE

- Frontend
  - Knows about user jobs and requests Glideins
  - "Constant pressure policy": keep a certain number of idle glideins all time
  - Operated by VO admins
- Factory
  - Knowns about sites and submit Glideins as requested by the frontend
  - Not a VO specific service
- Glidein
  - Validate environment (installed software, disk space, etc.)
  - Download, configure and run HTCondor daemon
  - Does cleanup at the end

GLIDEINWMS AT A GLANCE