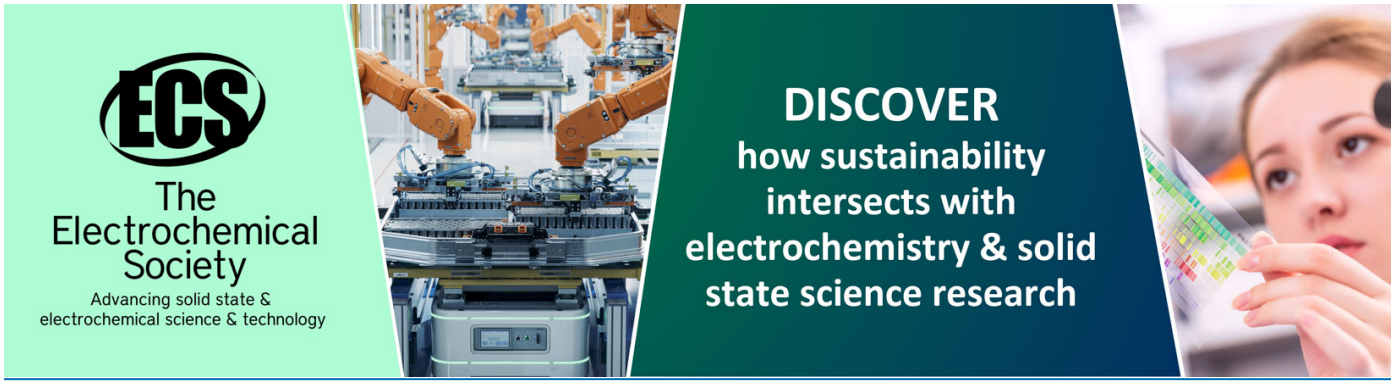
Interfacing HTCondor-CE with OpenStack



IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 092021 doi:10.1088/1742-6596/898/9/092021

Interfacing HTCondor-CE with OpenStack

B Bockelman[1], J Caballero Bejar[2], J Hover[2]

[1] University of Nebraska-Lincoln, Lincoln, NE 68588, USA
[2] Brookhaven National Laboratory, PO BOX 5000 Upton, NY 11973, USA

E-mail: bbockelm@cse.unl.edu, jcaballero@bnl.gov, jhover@bnl.gov

Abstract. Over the past few years, Grid Computing technologies have reached a high level of maturity. One key aspect of this success has been the development and adoption of newer Compute Elements to interface the external Grid users with local batch systems. These new Compute Elements allow for better handling of jobs requirements and a more precise management of diverse local resources.

However, despite this level of maturity, the Grid Computing world is lacking diversity in local execution platforms. As Grid Computing technologies have historically been driven by the needs of the High Energy Physics community, most resource providers run the platform (operating system version and architecture) that best suits the needs of their particular users.

In parallel, the development of virtualization and cloud technologies has accelerated recently, making available a variety of solutions, both commercial and academic, proprietary and open source. Virtualization facilitates performing computational tasks on platforms not available at most computing sites.

This work attempts to join the technologies, allowing users to interact with computing sites through one of the standard Computing Elements, HTCondor-CE, but running their jobs within VMs on a local cloud platform, OpenStack, when needed.

The system will re-route, in a transparent way, end user jobs into dynamically-launched VM worker nodes when they have requirements that cannot be satisfied by the static local batch system nodes. Also, once the automated mechanisms are in place, it becomes straightforward to allow an end user to invoke a custom Virtual Machine at the site. This will allow cloud resources to be used without requiring the user to establish a separate account. Both scenarios are described in this work.

1. Introduction

In parallel to the evolution of the Grid Computing technologies, a plethora of virtualization tools and techniques has been developed, both in industry and academia. Grid Computing has reached a high level of maturity in certain aspects, but it lacks some flexibility in others. In particular, the type of flexibility that virtualization tools can offer. For historical reasons Grid Computing technology has evolved around the needs of the High Energy Physics (HEP) community. This HEP community has a very limited set of computing requirements, but they are in fact almost unavoidable. One example of these contraints is the need for a specific Linux platform to be deployed on worker node resources within the HEP-oriented Grid infrastructure.

This scenario is highly efficient for the HEP experiments, but may present a barrier for other scientific communities.

On the other hand, many scientific and academia facilities offer to their employees -or even to external scientists-, the ability to use virtualization platforms. Several of these platforms have become very mature products over the past years. Some very popular examples are OpenNebula [1] and OpenStack [2]. The usage of these virtualized resources eliminates the contraints in the classic Grid infrastructure, but forces the users to learn how to use them, including installing client tools and getting virtualization-specific user accounts.

The goal of this work is to bring both worlds, the Grid Computing and the virtualization, closer to each other. We study how to allow users to submit their grid jobs to a standard Compute Element -the HTCondor-CE [3] in this work-, expressing platform requirements that do not necesarily fit the classical HEP-oriented environment. Also, we present a mechanism to let users to interact with a virtualization cluster without requiring special knowledge, the installation of virtualization-specific client software, or virtualization-specific user credentials.

1.1. Prototype setup

The setup for this prototype was simple. All grid identities were mapped at the CE as a single UNIX account, which corresponds to an unique OpenStack tenant. The back-end batch system was also HTCondor [4]. The VM images used in OpenStack had an HTCondor startd preconfigured to join the existing HTCondor pool. These startd daemons were also preconfigured to shut down after the first job finished, in order to prevent them from picking more than one job during this study. Also, job submission was done one by one.

The technical specifications of the OpenStack cluster used for this prototype are as listed:

- OpenStack instance: Icehouse.
- 120 compute nodes (16 cores, 32 GM RAM, 2 to 5 TB disks).
- 200 TB Swift (Amazon S3-equivalent) object store.
- EC2 API enabled.

The version of HTCondor was 8.4.8.

2. Prototype description

In order to allow the HTCondor-CE to decide when jobs need to be executed on a compute server within OpenStack instead of any of nodes in the back-end batch system, we make use of one feature included in the CE: the Job Router hooks [5].

The Job Router is, by definition, an add-on that transforms jobs from one type to another according to a configurable policy. The Job Router hooks are arbitrary code that can be invoked at some points during the job life cycle. Example on how to set HTCondor-CE to use hooks can be seen in Snippet 1.

2.1. Case 1: elastic expansion

The first scenario allows running user's job within OpenStack when the job's requirements do not match the characteristics of the physical nodes in the back-end batch system. A typical

```
JOB_ROUTER_HOOK_KEYWORD = NOVA

NOVA_HOOK_TRANSLATE_JOB = /usr/share/virtualgridsite/nova_hook_translate_job
NOVA_HOOK_UPDATE_JOB_INFO =
      /usr/share/virtualgridsite/nova_hook_update_job_info
NOVA_HOOK_JOB_EXIT = /usr/share/virtualgridsite/nova_hook_job_exit
NOVA_HOOK_JOB_CLEANUP = /usr/share/virtualgridsite/nova_hook_cleanup_job
```

Snippet 1: example of HTCondor-CE configuration for the Job Router hooks. virtualgridsite is the name of the package with this prototype.

case is when user's job requires and older version of the operating system, or a newer one. Table 1 shows a list of HTCondor classads defined for this purpose.

job classad	description
+opsys	Type of the OS. Example: LINUX
+opsysname	Name of OS. Example: CentOS
+opsysmajorversion	Version of the OS. Example: 7
+maxMemory	Amount of RAM memory needed.
+disk	Hard disk size
+xcount	Number of cores

Table 1: Job's classads for worker node requirements.

The process is shown in Figure 1. The HTCondor-CE's TRANSLATE hook compares the job's requirements with a set of configuration files compiling the entire list of available host profiles, both in the back-end batch system and VM images ready to be used in OpenStack. As a result of that comparison, when it decides a new VM-based worker is needed to run the job, it directly requests that OpenStack boots the appropriate VM. As mentioned, the VMs are prepared to initialize an HTCondor's startd daemon and join the back-end pool. Once the VM booting process has finished, the TRANSLATE hook finalizes and the source job gets routed to the back-end pool, with the guarantee that there is now a host that can execute it. It also added a new ad-hoc classad to the job with the name assigned to the VM worker.

The job execution finalization triggers a call to the CLEANUP hook. This hook will note the previously mentioned custom classad, and wil then proceed with the termination of that particular VM instance.

2.2. Case 2: interactive usage

In this case, the user knows there is an OpenStack cluster behind the CE, and she actually wants to boot a general-purpose VM host to log into and work interactively. In this case there is neither an HTCondor startd on the VM nor a payload to be executed.

This request for an interactive VM is expressed by adding a special job classad:

+virtualgridsite_interactive_vm = true

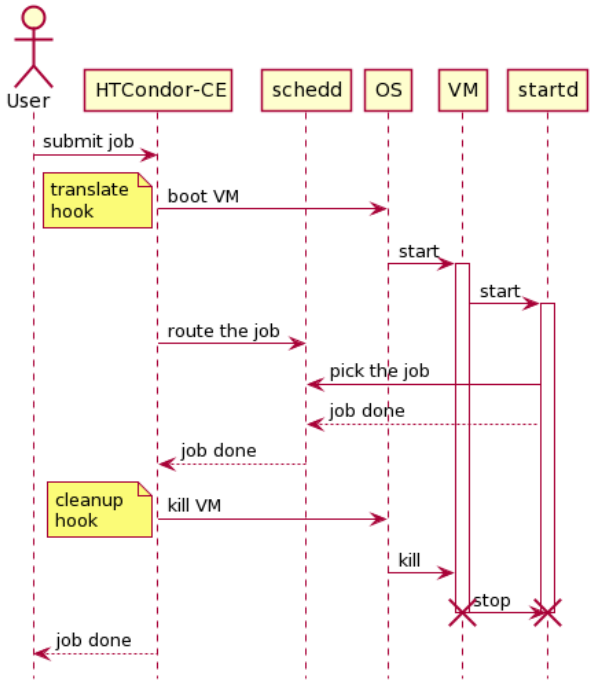


Figure 1: Sequence diagram for the elastic case. Solid lines represent actions, dotted lines represent information.

As can be seen in Figure 2, when the TRANSLATE hook detects that job classad, it transforms the job in place -so it is not routed to any back-end batch queue-, and it is converted into an EC2 job [6]. This is possible because, as mentioned in section 1.1, the OpenStack infrastructure has the EC2 API enabled.

In this scenario, it is the routed job, now converted into an EC2 job, the one in charge of requesting the VM instantiation in OpenStack. Once that step is completed, the user can read the public IP of that new VM server, and log into it. This IP is available because it is stored in a job classad set to be mirrored back with the setting in Snippet 2.

```
NOVA_ATTRS_TO_COPY = EC2ElasticIP
```

Snippet 2: Setup for a classad to be mirrored from the routed job to the source job.

Once the user no longer wants the VM, she can issue a *condor_rm* command, which will remove the running job. Once again, that event will be captured by the CLEANUP hook, which will execute code to terminate the corresponding VM instance.

2.3. Custom virtual machine image

Users can, if they prefer, provide their own VM images. This is done by passing an URL pointing at the image as a job classad:

+virtualgridsite_url = <URL with the VM image>

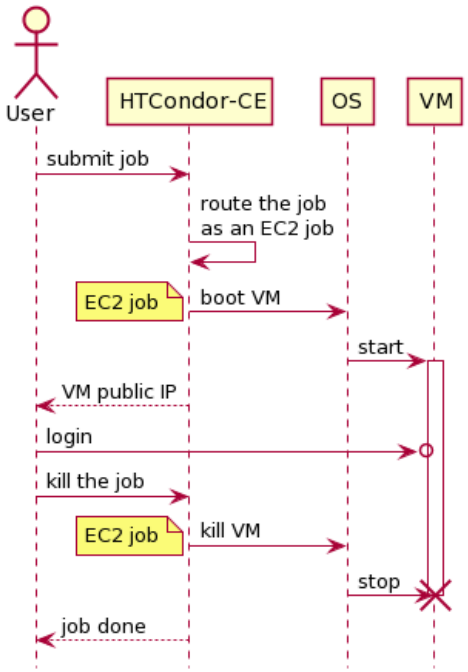


Figure 2: Sequence diagram for the interactive usage case. Solid lines represent actions, dotted lines represent information.

As can be seen in Figure 3, the only difference with respect the previous cases is that the new VM image is uploaded into Glance -the OpenStack image service- when needed. This image is uploaded with an unique name, created as a combination of the URL hash and its timestamp. This way it becomes trivial to determine whether the requested image has already being uploaded or not.

3. Security

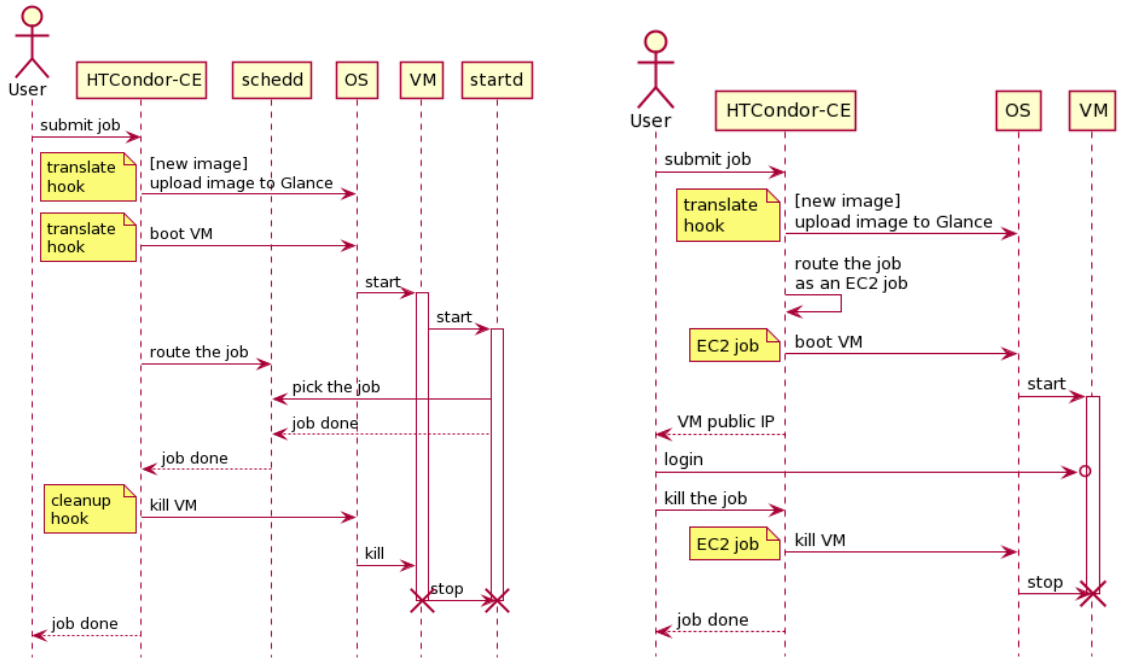
Because the configuration files, as well as the OpenStack credential files, are placed on the same CE host, it is important to prevent the user's jobs from running on that host. This is prevented by adding the configuration shown in Snippet 3.

```
START_LOCAL_UNIVERSE = False
START_SCHEDULER_UNIVERSE = $(START_LOCAL_UNIVERSE)
```

Snippet 3: Setup to prevent user's jobs being forked at the HTCondor-CE host.

Another security aspect involves the startd inside the VMs joining the HTCondor pool. As the prototype in this study was simple, with all components in an isolated environment unreachable from the outsite, a password-based security setup was enough. However, for more realistic on-production scenarios, this should be done more carefully.

From the point of view of the running jobs, there are no extra concerns with respect to security, traceability, etc. Once the node joined the HTCondor pool, it is indistinguishable from any other working node.



(a) elastic case (b) interactive usage

Figure 3: Sequence diagrams when using user's custom image. Solid lines represent actions, dotted lines represent information.

4. Problems found

Several limitations were found during this investigation. Some of these limitations are currently being fixed in the HTCondor source code. Deeper understanding of the others may lead to improvements in the overall job-routing system.

If the communication process with OpenStack by the TRANSLATE hook fails, there is not currently a clean way to terminate the job, or to put it in a HOLD state. Instead, the Job Router tries again, after 30 seconds, in an infinite loop.

For the mirroring of the classads from the routed job back to the source job, it is required that the UPDATE hook sends the whole classad to standard output. It has been found that action fails due to hidden bugs in the code.

It is not easy to manage a job when its requirements cannot be satisfied by any node in the back-end batch system nor any currently available VM image. The solution implemented, for the time being, is to detect that case early in the job cycle management, and create a dedicated route for it. To detect this case, we leverage a feature that allows for the partial creation of the routing table by code. This code reads the configuration files with all available image types, and builds the logic to identify jobs that do not meet any of those criteria. When this ocurrs, the job is transformed in place -and therefore not routed to the back-end batch queue-, with an extra classad (set_noreroute=True) to prevent it from being considered again for routing. This leaves the job in permanent IDLE status, making it a candidate for explicit removal. It can also be cleaned up automatically if a periodic_remove expression is added to its classad definition.

```
JOB_ROUTER_ENTRIES_CMD =
      /usr/lib/python2.6/site-packages/virtualgridsite/routes.py
JOB_ROUTER_ENTRIES_REFRESH = 600
```

Snippet 4: location of code with routing policies.

```
[ Name = "No_route";
EditJobInPlace = true;
set_noreroute = "True";
Requirements = TARGET.noreroute is undefined && <node requirements>;
set_PeriodicRemove = ( JobStatus == 1 && ( time() - EnteredCurrentStatus ) > 10 );
TargetUniverse = 5
]
```

Snippet 5: Example of routing table to manage jobs with requirements that cannot be satisfied. The classad noreroute will be detected, and used to prevent this job from being routed again.

Another problem found is related to the present configuration after installation rather than code design. The HTCondor-CE only allows, by default, jobs being routed in certain ways - more exactly, to certain Job Universes-. It is easy to workaround this constraint by overriding the configuration variable JOB_ROUTER_SOURCE_SOURCE_JOB_CONSTRAINT, removing that specific constraint, as shown in Snippet 6.

```
JOB_ROUTER_SOURCE_JOB_CONSTRAINT =
(target.x509userproxysubject =!= UNDEFINED) &&
(target.x509UserProxyExpiration =!= UNDEFINED) &&
(time() < target.x509UserProxyExpiration)
```

Snippet 6: overriding the default Job Router constrains

5. Future work

This work can be improved in several ways.

First, it should be straigthforward to expand the same concept to other platforms beyond OpenStack. For example, to AWS, or remote clusters using the BOSCO-CE mechanism [7].

The presetup in the VM images should be avoided, allowing the HTCondor startd to join any arbitrary pool.

The job lifecycle management allows for improvements. One possibility is to allow the same VM worker to run more than one job, increasing the efficiency of the whole system. Another option is to reuse the presented mechanism to elastically expand the size of the local back-end batch queue when the jobs waiting time passes some threshold.

Having more than just one OpenStack user (tenant) is highly recommended to allow the usage of user quotas, fair share mechanisms, and to enable per-tenant accounting and billing.

Removing the need for GSI authentication to submit jobs to the HTCondor-CE is under consideration.

We are also considering a change in the architectural design to prevent the hooks code from interacting directly with OpenStack. Instead, an independent daemon would listen to the hooks' requests and execute the corresponding steps. This will avoid the need to re-read the configuration files for each job, and also allows for having a single process which holds the state of the entire system, including all jobs. This centralization could facilitate the implementation of more complex and smarter policies.

Acknowledgments
The authors would like to thank the HTCondor developers for their constant support.

References

[1] Llorente I M, Moreno-Vozmediano R and Montero R S 2009 "Cloud computing for on-demand grid resource provisioning" *Advances in Parallel Computing, IOS Press* **Vol. 18** 177-191
[2] OpenStack LLC 2012 "OpenStack: The Open Source Cloud Operating System" *[Online]. Available: http://www.openstack.org/software/*
[3] Bockelman B, Cartwright T, Frey J, Fajardo E M, Lin B, Selmeci M, Tannenbaum T and Zvada M 2015 "Commissioning the HTCondor-CE for the Open Science Grid" *Journal of Physics: Conference Series* **664, No. 6** 062003
[4] Thain D, Tannenbaum T and Livny M 2005 "Distributed Computing in Practice: The Condor Experience" *Concurrency and Computation: Practice and Experience* **17, No. 2-4** 323-56
[5] *[Online documentation]. http://research.cs.wisc.edu/htcondor/manual/v8.4/4_4Hooks.html*
[6] *[Online documentation]. http://research.cs.wisc.edu/htcondor/manual/v8.4/5_3Grid_Universe.html#SECTION00636000000000*
[7] Weitzel D and Bockelman B 2017 "Contributing opportunistic resources to the grid with htcondor-ce-bosco" *To be published in Journal of Physics: Conference Series*