Journal of Physics: Conference Series **513** (2014) 032019 doi:10.1088/1742-6596/513/3/032019

Using the CMS High Level Trigger as a Cloud Resource

David Colling[1], Adam Huffman[1], Alison McCrae[2], Andrew Lahiff[3], Claudio Grandi[4], Mattia Cinquilli[2], Stephen Gowdy[2], Jose Antonio Coarasa[2], Anthony Tiradani[5], Wojciech Ozga[2], Olivier Chaze[2], Massimo Sgaravatto[6] and Daniela Bauer[1]

[1]Imperial College London, South Kensington Campus, London SW7 2AZ, UK
[2]CERN, CH-1211, Genève 23
[3]STFC, Rutherford Appleton Laboratory, Didcot, Oxfordshire, OX11 0QX, UK
[4]INFN Viale Berti-Pichat 6/2, I-40127 Bologna, Italy
[5]Fermilab, P.O. Box 500, Batavia, IL 60510-5011, USA
[6]INFN, Via Marzolo 8, 35131 Padova, Italy

E-mail: d.colling@imperial.ac.uk

Abstract. The CMS High Level Trigger is a compute farm of more than 10,000 cores. During data taking this resource is heavily used and is an integral part of the experiment's triggering system. However, outside of data taking periods this resource is largely unused. We describe why CMS wants to use the HLT as a cloud resource (outside of data taking periods) and how this has been achieved. In doing this we have turned a single-use cluster into an agile resource for CMS production computing.

While we are able to use the HLT as a production cloud resource, there is still considerable further work that CMS needs to carry out before this resource can be used with the desired agility. This report, therefore, represents a snapshot of this activity at the time of CHEP 2013.

1. Introduction

The CMS experiment at the LHC employs a multilevel triggering system[1]. The highest level of the trigger system is a large compute farm that uses algorithms very similar to those used in the offline system to make the decisions as to which data to reject and which data to write to tape. Despite having this very specialist role the High Level Trigger (HLT) is architecturally very similar to the general purpose clusters on which much of CMS' computing is performed. It is therefore not surprising that CMS would like to use this large cluster as a general purpose resource when the experiment is not taking data.

During the "first long shutdown" (LS1), from the end of 2012 to the beginning of 2015, the HLT is clearly very rarely required in its primary triggering role (only during a limited number of cosmic ray tests). However, CMS plan to be able to use the HLT as general resource during the shorter, week long, machine breaks and even possibly between fills of the LHC. There might even be some prospect of using parts of the HLT when, in triggering terms, CMS is operating in less challenging environments such as heavy ion running, although this is highly speculative at the moment.

The vital role that the HLT plays in data acquisition at CMS means that its use as a general purpose resource must *never* interfere with its primary triggering role. For this and other reasons CMS decided that the best methodology for using the HLT outside of data taking was as a cloud resource.

It was felt that a resource as large as the HLT should genuinely be general purpose when not part of the trigger system. Specifically it should not be restricted to running less I/O demanding workflows (such as Monte Carlo event generation). To this end it was decided that the benchmark workflow that would be used for testing the HLT in this mode was the re-reconstruction of data. This is one of the more challenging workflows required within the CMS computing system.

An initial configuration capable of carrying out reprocessing was designed and implemented. The testing of this initial set up was very instructive and it became clear that several modifications would be required in order to make the HLT usable for data re-reconstruction at an appropriate scale.

These modifications have been made and the HLT is currently acting as a general purpose resource (carrying out the reprocessing of 2011 data) within CMS. However, constraints still remain and much work is still to be carried out for the HLT to be usable at its full potential.

2. The CMS Trigger System

The architecture of the CMS Data Acquisition (DAQ) System is shown in Figure 1. The top of this figure is closest to the detector electronics and the offline computing services are at the bottom of the figure, with data flowing from the top to the bottom. There are two levels of triggering employed in the CMS DAQ, the Level 1 Trigger and the HLT. The level 1 trigger is built from dedicated hardware and this takes the event rate down from the (up to) 40 MHz of the LHC collisions to 10^5 Hz. Events passing the Level 1 Trigger are passed on to the HLT (which is the light blue oval labeled "Filter System" in Figure 1), where a finer selection takes the rate down to a few $\times$ 10^2 Hz. The HLT is a large compute cluster built from commodity hardware located near to the CMS experiment.

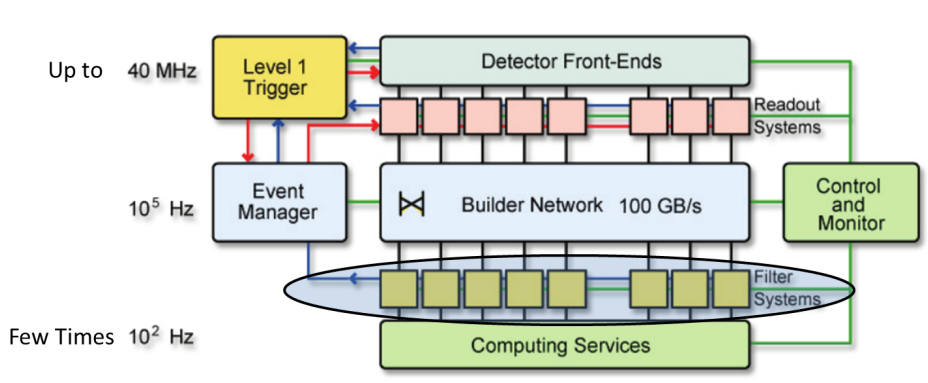

Figure 1. The CMS DAQ system. At the top of the figure are the detector readout electronics. These feed into the Level 1 trigger, which in turn determine whether or not the event is readout from the rest of the detectors. If the detector are readout their output is put togethr to form an event which is then passed to the HLT (the blue oval labelled "Filter system") for further processing. Finally, the HLT makes the decision as to whether the event is written to tape or discarded. The frequencies at the side of the figure show the rates at which decisions are being made. (Modified from original in [2])

Data which passes the HLT selection is written to tape at the Tier 0 located on the main CERN site, some kilometers from the CMS experiment.

2.1. The HLT at the end of 2012

While the HLT was built from commodity hardware it was designed solely to be a trigger farm. This meant that much of the configuration was different from that normally expected to be found on a general purpose cluster built from a similar level of hardware. Some of the characteristics of the HLT as it existed at the end of 2012 included:

- The HLT is *wholly* owned by CMS
- Apart from the system disks in the machines themselves, there was essentially no storage associated with the HLT.
- There was very little monitoring of the HLT.
- The HLT was linked to CERN via two different network paths. One of these was at 2×10 Gb/s and the other was at 1 Gb/s. The network configuration was such that *all* traffic, apart from the experimental data flowing from the detector to CERN, was carried by the single Gigabit connection.
- The cluster was built from three different generations of machine, with widely different specifications.

The total size of the HLT at the end of 2012 was approximately 195kHEPSpec06 [3] and this is predicted to grow by the restart of the LHC in 2015. This is comparable with the *total* CMS tier 1 requirement. Clearly, a resource of this scale must be fully utilised.

3. Using the HLT as a General purpose resource and the choice of cloud infrastructure

The requirement that other uses of the HLT must *never* impinge on its use as a trigger farm lead directly to further constraints which include:

- Online Computing must retain control.
- Only "spare" resources are available to Offline computing.
- Non-trigger use must make minimal changes to the underlying hardware.
- Offline computing must be able to migrate on and (especially) off at very short notice (15 minutes cited) if there is any hope to use the HLT between fills.

These constraints suggested a cloud architecture that could be "overlaid" on top of the HLT hardware when it was not in use as a trigger. Such a layer of virtualisation also provides a good mechanism for utilising sets of heterogeneous hardware resources as the virtual machines (VMs) will only start on physical nodes capable of supporting them.

3.1. Why choose OpenStack

Several cloud architectures were discussed. OpenStack[4] was chosen because:

- It has a large, growing and dynamic development community.
- It is Open Source.
- It is widely believed to have a solid underlying architecture
- It is compatible with the EC2 API
- It is being driven by a huge and growing user community [1]

In the beginning of 2013 OpenStack ("Essex" edition) was deployed across the HLT to form the *CMS openstack, opportunistic, overlay, online-cluster Cloud* (CMSooooCloud).

[1] "People Get Pissed Off About OpenStack. And Thats Why It Will Survive" [5]

4. The CMSooooCloud Initial Configuration

When designing the initial configuration of the CMSooooCloud it was decided that this resources needed to be as versatile as possible and that it should be capable of carrying out a wide range of CMS computing tasks. Data reprocessing was chosen as an exemplar workflow that would operate all the functionality required to make CMSooooCloud a generally useful resource. However, it was also decided that it was more important to produce a functioning prototype than it was to try to design a perfect system from scratch. This was because of the lack of maturity (at that stage) of some of the components in the system and hence a lack of understanding as to how they might scale. Lessons could then be learnt from this initial configuration and applied to future configurations. Crucially, this meant that in the initial configuration *all* network traffic between the CMSooooCloud and CERN went over the 1 Gb/s link.

We also introduced some basic monitoring into the CMSooooCloud so that we could asses network traffic, machine load etc. This monitoring was still significantly less than that which would normally accompany a cluster of this size.

In order to carry out data reprocessing it was clear that CMSooooCloud would need to interact with a range of services.

4.1. Data Access

In order to reprocess data, CMSooooCloud would need to be able to read in the original data files and to write out the reprocessed data. In the traditional Grid data access model used by CMS, the lack of any sizable disk resources at the CMSooooCloud would have been a major data access problem. However, the development of the CMS AAA system [6], which allows remote read (and write) access to CMS data over the network removed this constraint. Specifically, we decided to read the original data from and write the reprocessed data to the EOS[7] file system at CERN over xrootd[8].

4.2. Conditions Data Access

This was initially considered not to be a problem as CMS already served conditions data over the FroNTier service[10] based on squid caches. So all that was needed was to configure the VMs to point at a FroNTier server at CERN.

4.3. CMS Software

CMS specific jobs, such as data reprocessing, clearly need access to the CMS software. Two approaches were considered. The first was to build releases of the CMS software into the virtual image to be run on CMSooooCloud. However, this would mean that every time there was a new release of the CMS software a new image would have to be produced. So instead it was decided that the CMS software should be exported to the VMs running on the CMSooooCloud via CvmFS[11] which is a software distribution system based on squid caches. This approach allows the VMs access to all the releases of CMS software being served over CvmFS.

4.4. Job Submission Mechanism

The GlideinWMS[12] is CMS' production workload management system.

GlideinWMS is a pilot-based workload management system that is built on top of HTCondor[13]. Functionally, it can be organized into following logical architectural entities:

(i) GlideinWMS Provisioning Service that consists of Glidein factory and VO Frontend services:
- VO frontend monitors the status of jobs in the HTCondor Scheduler and regulate the number of pilot job requests made to the factory.
- Glidein factory that acts on the frontend request and submits pilot jobs to the grid/cloud sites using HTCondor-G[14][13] as a grid submission tool.

- HTCondor collector as a communication dashboard used for message exchange between different GlideinWMS services.

(ii) Glidein Service, otherwise known as a glidein pilot is a lightweight job that runs on the worker node and bootstraps the environment on the worker node to run actual user jobs.

For Grid computing, this means that the GlideinWMS Provisioning Service submits the Glidein Service as a job to a remote Grid Site. The Grid Site is responsible for administration of the batch system that the Glidein Service runs on. Access to remote Grid Sites is controlled via x509 credentials. The same x509 credentials can be used to access multiple Grid Site computing facilities.

Running on Cloud Resources required conceptual changes as well as a couple of architectural changes[13] Conceptually, the Glidein Service, or pilot, is no longer the lightweight job that is submitted to a remote batch system. The Glidein Service is now a virtual machine request. The virtual machine image must be pre-configured to bootstrap the Glidein Service and pre-staged to the Cloud. Another conceptual change is that administration of the "worker node" has shifted from the remote Grid Site administrators to the operators of the glideinWMS infrastructure.

Architecturally, glideinWMS had to be modified to handle multiple types of credentials now. Most Cloud providers do not support x509 credentials. Instead, they provide their own credentials in the form of an Access Key and a Secret Key. The runtime behaviour had to be adjusted in order to prevent certain Cloud implementations from being overwhelmed.

In the end, the researcher using glideinWMS to get their work done will not notice a difference.

5. Experiences with the CMSooooCloud Initial Configuration

Initial testing of the initial infrastructure configuration threw up a range of minor, yet annoying functional problems. These reflected the evolving nature of the software components being used and that many of them were being in an environment that was different from that for which they had originally been developed.

A list of these minor problems would include:

- Permissions problems with xrootd and EOS
- VMs dying because access to CvmFS was not available fast enough
- OpenStack EC2 not Amazon EC2 causing many minor problems all of which required modifications to the glideinWMS.
- Behaviour in clouds is different from behaviour in Grids so glideinWMS needed to learn how to handle the situations differently
- OpenStack controller can be "rather fragile" when asked to do things at scale so glideWMS learnt to treat it gently.
- glideinWMS losing track of jobs (often through fragility of OpenStack) and jobs ending up in "shutoff" state
- ...

Gradually these functional problems were resolved and the system was shown to work well at a small scale. However, scaling tests showed there to be more fundamental problems with the initial configuration.

The first fundamental problem that we encountered while scaling the testing of the CMSooooCloud was that prepossessing jobs started to fail if the total number of running jobs exceeded approximately a thousand jobs. This can be seen clearly in Figure 2.

Investigation of this effect showed that it was due to the limitations of the 1 Gb/s link over which all the data being reprocessed was transported. More detailed investigation showed that the network link had been a bottleneck for the reprocessing for much smaller numbers of jobs.

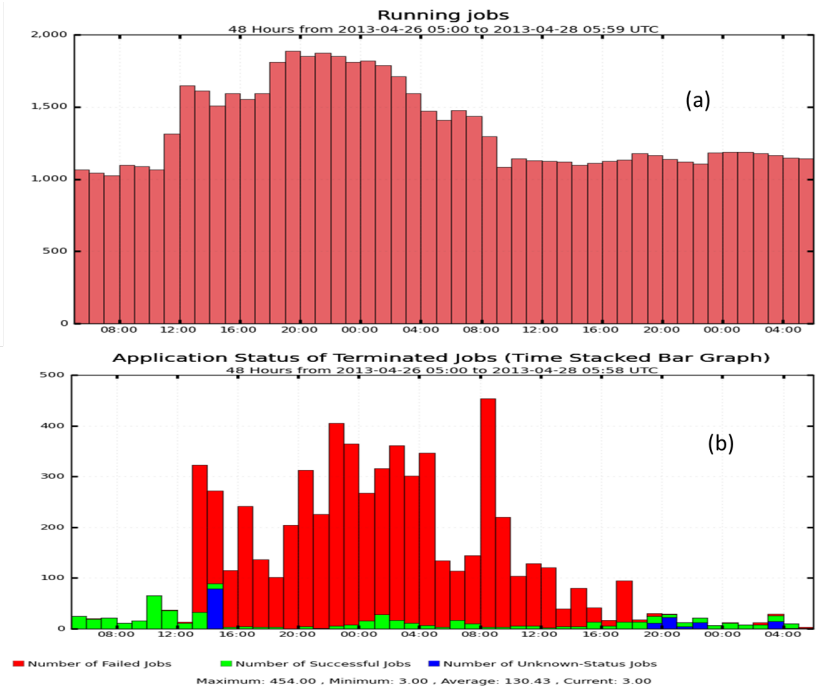

Figure 2. Plots showing (a) the number of jobs currently running and (b) the success/failure of jobs completing

These jobs had run (very) inefficiently, however it was only when number of concurrently running jobs reached about a thousand that the contention on the link was sufficient to cause the jobs to time out en masse. It was decided to reconfigure network so that all traffic associated with CMSooooCloud went over the 2×10 Gb/s network link to CERN. This significantly reduced this bottleneck.

After the reconfiguration had reduced network contention, jobs failure rates still grew as their number went above about a thousand, although far less dramatically. Investigation showed this to be caused by overloading of the squid servers used by all the VMs. In order to alleviate this bottleneck a squid server was installed on each hypervisor. These were in turn cascade to (at least) two other instances until gradually percolating to the original server. In This way the load is spread throughout the cluster.

6. Current Configuration and operation

The CMSooooCloud has recently been upgraded to the latest OpenStack version (Grizzly). All network traffic between CMSooooCloud and CERN goes over the 2×10 Gb/s network link and the squid caches have been arranged as described. The CMSooooCloud is running as a production resource and Figure 3 shows, as an example, the successful running of a reprocessing workflow on CMSooooCloud and the associated traffic on the network to CERN (labeled as the CDR network) on the day before this paper was compiled.

7. Ongoing Work

The CMSooooCloud is a production resource, however there are several areas in which further development is needed if it is to fulfill its intended role in full. The three most obvious areas where development is needed are:

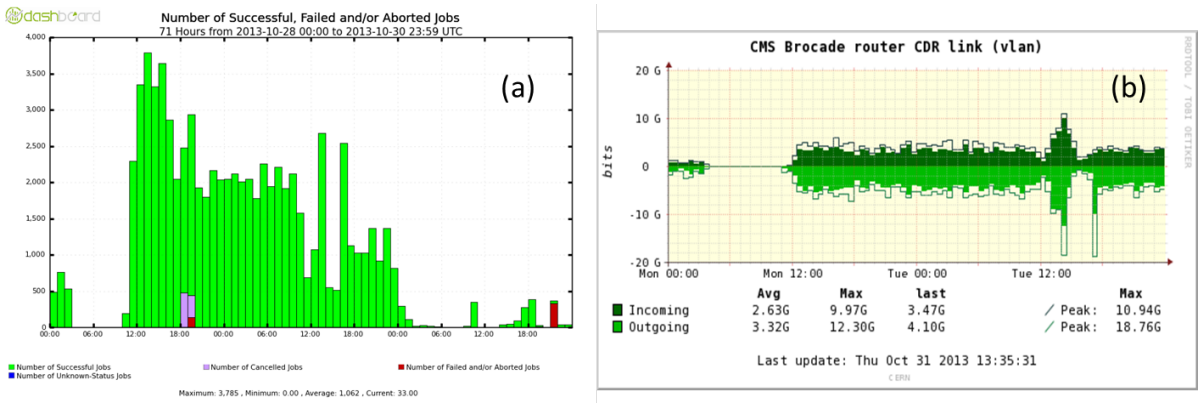

Figure 3. Plots showing (a) the number of jobs completing as success/failure and (b) the network traffic over the 2×10 Gb/s (labeled CDR) network link between the HLT and CERN.

- The network will need to be upgraded. Currently it can sustain a (few) thousand jobs, but will need to be able to run 15000 jobs concurrently by 2015
- Increasing monitoring of VMs themselves (especially) and other parts of the infrastructure.
- Migration strategies so that CMSooooCloud is capable of running in the short gaps between data taking.

This activity takes place between different communities within CMS and the communication between them is as important to the successful use of CMSooooCloud as many of the technical issues.

8. Conclusions

CMS have overlaid an OpenStack cloud onto their HLT farm to form the CMSooooCloud. This should provide both the functionality and agility required to use the HLT as a more general purpose resource outside of data taking periods. While the CMSooooCloud is being used as a production resource, there is much work still to do if it is to realise its full potential.

References

[1] (accessed on Oct 31, 2013): http://cms.web.cern.ch/news/triggering-and-data-acquisition
[2] CMS, TriDAS Project, Technical Design Report, Volume 2: Data Acquisition and High Level Trigger *CERN/LHCC 02-26*
[3] (accessed on Oct 31, 2013): https://twiki.cern.ch/twiki/bin/view/FIOgroup/TsiBenchHEPSPEC
[4] (accessed on Oct 30, 2013): http://www.openstack.org/
[5] (accessed on Oct 14, 2013): http://techcrunch.com/2012/08/13/people-get-pissed-off-about-openstack-and-thats-why-it-will-survive/
[6] (accessed on Oct 31, 2013): http://indico.cern.ch/getFile.py/access?contribId=681&sessionId=78&resId=0&materialId=slides&confId=181298
[7] (accessed on Oct 31, 2013): http://information-technology.web.cern.ch/services/eos-service/full
[8] (accessed on Oct 31, 2013): http://xrootd.slac.stanford.edu/
[9] (accessed on Oct 31, 2013): http://frontier.cern.ch/cmsarchitecture.html
[10] CMS conditions data access using FroNTier. By USCMS Collaboration (Barry Blumenfeld et al.). Prepared for International Conference on Computing in High Energy and Nuclear Physics (CHEP 07), Victoria, BC, Canada, 2-7 Sep 2007. Published in J.Phys.Conf.Ser.119:072007,2008.
[11] (accessed on Oct 14, 2013): http://cernvm.cern.ch/portal/filesystem
[12] GlideinWMS homepage: (accessed on Oct 14, 2013): http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS
[13] HTCondor Week 2013 Talk: http://research.cs.wisc.edu/htcondor/HTCondorWeek2013/presentations/TiradaniA GlideinWMS.pdf [14] Frey J et al. 2002 Condor-G: A Computation Management Agent for Multi-Institutional Grids Journal of Cluster Computing vol 5 pp 237-246