Backfilling a cloud with grid jobs

Simon Fayer[1,*] and *Dan* Whitehouse[1]

[1]Blackett Laboratory, Department of Physics, Imperial College London, London SW7 2AZ, United Kingdom

Abstract.
Imperial College London hosts a large Tier-2 WLCG grid site based around a HTCondor batch system; additionally it provides cloud computing facilities using OpenStack to non-WLCG activities. These cloud resources are open to opportunistic usage provided the impact on the primary cloud users remains low.
In common with most Tier-2 sites we see constant job pressure from the WLCG VOs, while the usage of our cloud is much more intermittent. To allocate grid jobs to the available opportunistic cloud resources we implemented a lightweight backfill system based on the OpenStack Python API.

1 Introduction

The Imperial College Tier-2 WLCG grid site is a large Tier-2 site with approximately 16000 job slots hosted on dedicated bare metal worker nodes running CentOS-7. It provides compute and storage resources to three WLCG experiments (CMS, LHCb and ATLAS) and a number of non-WLCG communities in the neutrino, dark matter and astrophysics sectors.

In addition to its traditional grid compute and storage resources, the site also hosts a private cloud with a capacity of approximately 4000 job slots, serving a limited number of non-WLCG communities. The usage of this cloud is intermittent as is typical for a setup where large processing campaigns are followed by a period of analysis and development. Conversely, the job pressure on the grid site is fairly constant and with few exceptions the demand for job slots exceeds the available resources. We use a lightweight, custom-built backfill system to allocate spare cloud resources to the grid site with minimal impact on our cloud users, which we describe below.

2 Overview

The technologies used for the grid site compute cluster are a HTCondor[1] batch system with HTCondorCE frontends and OpenStack[2] for the cloud. The OpenStack system comprises of 55 hypervisors providing a total of approximately 4000 CPU cores. OpenStack is configured to use the local hypervisor disk as the backing store for the instance disks to improve local I/O performance. The instances are allocated private IP addresses and connect to the HTCondor pool via a network address translation (NAT) router.

*e-mail: lcg-site-admin@imperial.ac.uk

The aim of this project is to make use of unused capacity in our private cloud to supplement our grid site's batch system. This allows us to ensure we can have enough resources and flexibility in our cloud for large temporary projects, but not have nodes sat idle when these projects are not running. The solution needs to be able to release resources back for dedicated cloud use quickly when there is demand for them.

The backfill system consists of simple Python scripts built on the OpenStack application programming interface (API). For the cloud instances we use custom images that contain all of the necessary packages and configuration required by grid-type workloads. We begin by describing the method and infrastructure used to build these images. We then discuss the scripts used to manage these instances and finally we will cover how the system is monitored using Ganglia.

3 The Image Build System

We make use of the Jenkins[3] automation server to run the image building pipeline on a bare metal server. While most of our services run in Virtual Machines (VMs) using KVM[4], using nested virtualisation for the image build system resulted in significantly increased build times, slowing down development. To build our images, we use the *Oz*[5] image creation software. Our build system comprises of the *Oz* configuration, a kickstart script for RHEL based distributions, configuration templates for the virtual machine image and installation scripts; these are stored in a git repository[6]. We include an entry script executed by Jenkins which builds the virtual machine image using KVM and the OpenStack client applications, in order to upload the completed image to our OpenStack Glance server. HTCondor host certificates and keys, as well as OpenStack credentials are secured using the Jenkins built-in secret handling system. To ensure that the operating system of the images stays current and that any security patches are applied promptly, we have configured Jenkins to build and upload the image nightly. We additionally make use of Github webhooks to initiate a build any time we push to the repository.

4 Build Description

We are currently using CentOS-7 as the base operating system for our image; this is installed in the virtual machine image using *Oz* and a kickstart script. We carry out a minimal install including *wget*, *vim* (for debugging), *autofs*, *HTCondor* and various packages to assist with provisioning on OpenStack (e.g. *cloud-init*). Using the *Oz* functionality to copy files into the image we deploy the following: SSH keys, kernel tuning parameters (sysctl.conf), HTCondor configuration, our post installation script, CVMFS configuration, and a script to automatically drain the node. Once all of the configuration is deployed we execute our post installation script; this sets all of the correct permissions on configuration files and installs and sets up CVMFS, it also creates job-slot specific users for HTCondor. "HEP_OSlibs"[7], a meta-package that captures the Linux OS dependencies for the software of the four LHC experiments, is also installed by the post install script.

5 Backfill algorithm

All jobs allocated via the backfill algorithm run in a dedicated OpenStack "Backfill" project, with its own private network and flavour. The number of vCPUs allocated to this project is determined by a script ("resources.py")[8] deployed on the OpenStack administration node and run as a cron job every 10 minutes. It determines the current spare capacity on the cloud

and allocates 90% of it to the "Backfill" project, we leave 10% as headroom for immediate use by our cloud users. We define a "Backfill" flavour to match the standard requirements for a typical grid job, with 8 vCPUs and 32GB of RAM. The script then iterates over all of the OpenStack hypervisors which are online and checks what the spare capacity is in terms of Backfill instances. The result is used to set the Backfill project quota, and any changes are logged for debugging.

A second script ("instances.py")[8] runs 5 minutes after the first script. It uses the OpenStack API to get the number of instances which can fit in the "Backfill" project based on the quota set by the resources script. It then checks how many existing instances there are and thus how many new ones can be created. Instances which have been shutdown, are in an "error" state, or have been stuck in a "build" state for over 3 hours are removed. Once things have been tidied up the script proceeds to create the calculated number of instances; finally metrics are calculated and posted to Ganglia. The following metrics are collected: current instances, failed instances, stuck instances, current quota and the number of instances added/removed in this run cycle.

Individual instances within the project are using a 60 minute time window to wait for jobs. If no jobs arrive within that time frame or if all jobs have finished *and* the instance is outside the 60 min time window, it is drained and shut down. We achieve this by setting up systemd services for our script to drain the node which creates a drain file 60 minutes after the virtual machine has booted. A cron job shuts the instance down once the drain file has been created and there are no longer any running jobs.

6 Conclusion

We have now been working with our backfill scripts in production for well over a year. Figure 1 shows a plot of running instances versus "target" instances, where the latter is defined as the number of instances which the script aims to have running based on the available capacity. While the scripts required some initial fine tuning, we now have a system in place which scales appropriately and responsively. We can see the algorithm working correctly from the figure as the red "Running Instances" line quickly changes to match the blue "Target Instances" line as it changes. The amount of additional resource we get from backfill varies depending on how the cloud is being used, but as an example at the time of writing, backfill is contributing over 2000 job slots to our HTCondor cluster which would otherwise be sitting idle therefore helping us to get the most out of the financial investment in providing our private cloud.

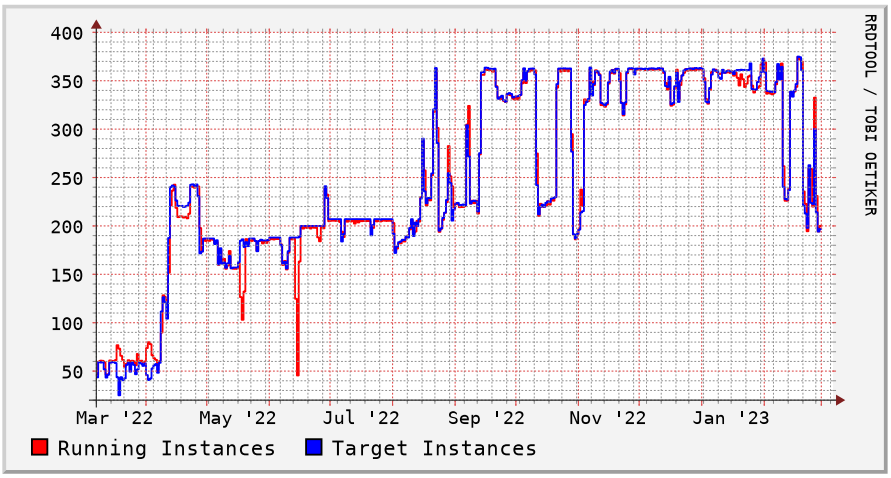

Figure 1. Running instances vs available ("target") instances, new hypervisor nodes were added in August 2022. The dips in the running instances line in May & June 2022 are when the backfill instances were halted for maintenance.

References

[1] HTCondor Team, https://htcondor.org, [accessed 2023-06-07]
[2] OpenStack, *Open Source Cloud Computing Infrastructure*, https://www.openstack.org, [accessed 2023-06-07]
[3] The Jenkins Project, https://www.jenkins.io, [accessed 2023-06-07]
[4] KVM Contributors, https://www.linux-kvm.org/, [accessed 2023-06-09]
[5] C. Lalancette, *Oz*, https://github.com/clalancette/oz, [accessed 2023-06-09]
[6] D. Whitehouse, S. Fayer, https://github.com/ic-hep/ichep-wn-image, [accessed 2023-06-16]
[7] CERN, https://gitlab.cern.ch/linuxsupport/rpms/HEP_OSlibs, [accessed 2023-06-16]
[8] D. Whitehouse, S. Fayer, https://github.com/ic-hep/openstack-scripts, [accessed 2023-06-16]