Using Apache VCL and OpenStack to provide a Virtual Computing Lab

Curtis Collicutt and Cameron Mann
Cybera
2-59C Computing Science Centre
University of Alberta
Edmonton, Alberta, T6G 2E8
{curtis.collicutt, cameron.mann}@cybera.ca

***Abstract*—Post-secondary institutions are looking for ways to provide virtual computing labs to students. Using the open source systems Apache VCL and OpenStack, as well as a pre-existing OpenStack provisioning module for Apache VCL, we implemented a pilot project to provide virtual computing labs. As the pilot continued we altered and improved our implementation and discovered new requirements. These improvements and lessons learned are presented here.**

Keywords*—*Apache VCL, OpenStack, Virtual Computing Lab

I. Introduction

Virtual desktop infrastructure (VDI) is an attractive solution to organizations that need to support large numbers of physical desktops or computing labs with increasingly smaller budgets and fewer staff. Some organizations, particularly post-secondary institutions, are searching for a relatively inexpensive solution for virtual desktop infrastructure (VDI) to either replace or enhance traditional computing labs.

A. Enterprise VDI

Typically, enterprise VDI solutions are expensive and in order to support input/ouput operations per second (IOPS) intensive workloads such as Windows 7 virtual machines, require considerable investment in backend storage technologies. Further, VDI deployments are often on a much larger scale than virtualized servers–for example, 500 virtual servers is a relatively large deployment, but 500 virtual desktops is not. Complicating the issue is that organizations often try to expand their existing enterprise server virtualization system in order to support VDI, but the workload associated with VDI requires a different architecture than the average enterprise virtualization deployment.

B. Virtual Computing Lab Requirements

Virtual computing labs do not necessarily have the same requirements as enterprise VDI systems. In our deployment of Apache VCL [1] the virtual machines are stateless and thus do not require persistent storage. Further, physical computer labs are not usually 100% utilized. In fact it is not unusual to find a ten or twenty-to-one ratio in terms of total Apache VCL system users to concurrent users. In cases of 100% concurrent instance utilization students can make reservations for future time slots via the scheduling system provided by Apache VCL. IPv4 exhaustion is also an issue, and until IPv6 is well supported, network address translation (NAT) seems like the only way to reduce IPv4 usage and still provide remote access to a large number of virtual desktops. Finally, users usually access virtual computing labs from their own devices instead of a thin desktop or other similar device.

C. Need for VDI-lite

The above issues, requirements, and use-cases suggest that there is a need for a VDI system that is somewhere between a relatively plain virtualization platform, such as a bare hypervisor, and a full blown Enterprise VDI system–something that we have been calling "VDI-lite."

II. Background

Apache VCL is an open source cloud computing platform that can provision virtual machines, provide time-based reservations, and broker remote sessions. OpenStack [2] is open source software used to create public and private clouds, including infrastructure as a service functionality that can be utilized by Apache VCL.

A. Pilot Project

Part of Cybera's mandate as an organization is to explore and apply new technologies. In previous work, Cybera had investigated enabling Apache VCL to provision virtual machines using the Amazon Web Services Elastic Compute Cloud (AWS EC2). OpenStack can operate in a similar fashion to EC2, and in fact supplies API compatibility with core AWS services. Given that Cybera employs OpenStack in other projects, we resolved to determine whether using OpenStack as the underlying provisioning system for Apache VCL is appropriate, performant, and cost effective.

Cybera worked with the University of Alberta and other educational institutions in Alberta to implement the pilot which provided virtual computing lab resources to students. Over the course of the pilot, Apache VCL provisioned thousands of reservations and virtual machines to hundreds of users, and also managed terabytes of environment images (Table I and Figure 1). Students were provided access to a wide variety of application software, from Matlab to Photoshop to ArcGIS, among others.

TABLE I. VCL USAGE - DEC 2012 TO APRIL 2014

VCL Usage Statistics	
OpenStack Instances Created	5995
VCL Images	75
Reservations	4386
Hours Used	9979
"Now" reservations	4206
"Later" reservations	185
Load times less than 2 minutes	3641
Load times greater than 2 minutes	745
Total Users	944
Course Statistics	
Courses offered using VCL	31
Students enrolled in courses	1480

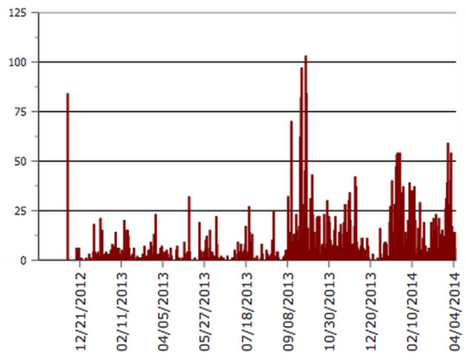

Fig. 1. Reservations by Day

B. *OpenStack Apache VCL Module*

We used the OpenStack module originally developed by Young-Hyun Oh [3] as the foundation for our VCL installation. As we gained experience using the module in a production system we made modifications to improve its reliability, including switching to a Perl SDK [4] to provide bindings to the OpenStack Compute API. The updated OpenStack module will be contributed back to Apache VCL for use by other interested groups. Other challenges included the overlap in provisioning tasks that OpenStack and VCL perform and the accommodations that must be made to avoid interference. For example, VCL expects its virtual machines to always have the same IP addresses while instances created in OpenStack have a range of possible addresses. This was a recurring problem that took multiple attempts to solve successfully.

C. *OpenStack Deployment*

The OpenStack deployment for our pilot project was only used by the Apache VCL system. While we did support multiple post-secondary institutions with this single "cloud," VCL was its only tenant. Most OpenStack clouds will support many tenants, and in fact, future work for our pilot as it moves into a new phase is to run within a larger OpenStack cloud with multiple tenants.

Cybera installed OpenStack Essex on a system comprised of eight Dell PE C6220 servers in two C6000 chassis. Each server had two Xeon E5-2650 2GHz processors and 128GB of RAM. One node was used as the OpenStack management node with the remaining seven set up with commodity solid state drives in a RAID 0 configuration. A Cisco 2960 was used as a management switch and an Arista 7050 as the top of rack switch.

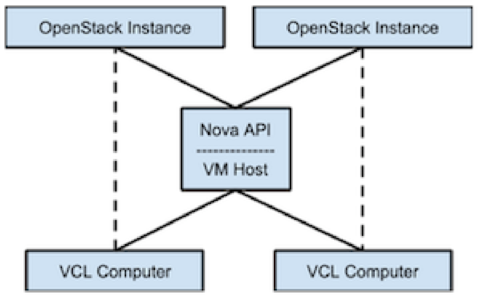

Fig. 2. Mapping VCL Computers to OpenStack Instances

III. OPENSTACK PROVISIONING MODULE

To run VCL on OpenStack, an appropriate provisioning module was required. We had previously started development of an EC2 provisioning module with the hope of supporting multiple platforms, however incomplete implementation of the EC2 API in OpenStack meant this was not a feasible solution. It was around this time that Young-Hyun Oh posted his prototype OpenStack provisioning module using the Nova API to the VCL mailing list [5]. The module had all the functionality we needed despite not configuring public access to any instances it created. Public access was instead enabled through a NAT patch we created since we did not have enough IP addresses to assign one to each instance. However, because the instances are not publicly accessible without NAT, the management node must be run as an instance within the same OpenStack tenant.

In terms of integration with VCL, a VM host must be configured for the provisioning module. However, this host does not correspond to any physical machine, but instead should be thought of as representing the Nova API endpoint of the OpenStack cloud. Virtual machine computers must then be created in VCL and added to that VM host in the same way as usual, though the MAC and IP address fields are ignored. The provisioning module then creates and maps OpenStack instances to VCL computers when reservations are created. In Figure 2 we can see how the VM host bridges VCL and OpenStack.

Since adopting the OpenStack module we have made a number of changes. The major recurring issue we ran into revolved around VCL's use of the hosts file to map the names of computers in VCL to their IP addresses. The first way VCL uses the hosts file is for all remote commands which are executed using the computer name and thus must be mapped to an IP address. The second way is to look up a computer's IP address. VCL first checks to see if the IP address has already been retrieved and is in memory; if not the hosts file is checked next and finally the database if it is not present in the hosts file. This behaviour depends on a static DHCP configuration such that each VCL computer receives the same private IP address every time. However, OpenStack uses dynamic DHCP, meaning that instances will not always receive the same private IP address. The original OpenStack module already handled this by dynamically updating the hosts file when an instance was created or deleted. It is worth noting that in our original EC2 provisioning module we also used this approach.

We started to see a number of duplicate entries where multiple computers would have the same private IP address, incorrect entries, and missing entries. The main symptom that resulted was multiple users could end up reserving the same computer and competing for access. In a very rare failure case this caused the management node to delete itself when VCL incorrectly identified its own instance as belonging to a reserved computer. This could happen because VCL would retrieve an incorrect IP address from the hosts file and use it to look up the incorrect instance in OpenStack. Our initial fix was to add a database table to map VCL computers to OpenStack instances to guarantee uniqueness and avoid identifying the wrong instance. We still, however, updated the hosts file for compatibility as that is how the core VCL code retrieves the private IP address for a computer.

Despite adding the computing mapping table the issue of missing IP addresses in the hosts file continued, though remote commands to those instances still completed. It was at this point we realized that OpenStack creates DNS entries for all instances based on their hostname, which matches the computer name in VCL. This meant that even with missing entries in the hosts file, VCL would be able to contact those instances. However, VCL would still fail when trying to retrieve those private addresses from the hosts file. We decided that rather than trying to manage the hosts file through the provisioning module, the better approach would be to modify core VCL code. We removed the code in VCL that checks the hosts file and replaced it with a DNS lookup. Since this fix we have experienced no further issues. However, further changes are still needed to integrate our code upstream as this fix will break VCL on most other platforms, including even other OpenStack clouds with different network configurations.

This is one area where we believe that changes to VCL are required to provide production ready OpenStack support, or any similar platform. Dependency on static resources like the hosts file conflicts with the dynamic nature of the cloud. There are a number of possible solutions to this issue, but the one that we favor is for VCL to query the provisioning module for the private IP address before checking the hosts file. This solution was suggested as a possibility on a recent VCL Office Hours call and would maintain backwards compatibility while allowing VCL to handle environments where the networking is less static than expected on a case-by-case basis.

IV. Network Address Translation

We required NAT support because we did not have enough public IP addresses to assign one to each virtual machine. We had already developed and deployed a NAT patch before transitioning to OpenStack, however some additional changes were necessary to support the new environment.

The NAT patch maps ports on the VCL management node's public interface to ports on the private interfaces of the virtual machines. For this to work in OpenStack it requires the management node to be virtualized and running in the same tenant as the instances it provisions so that it can access their private interfaces. Additionally, when running in OpenStack the management node will not have it's own public interface. Instead, OpenStack provides a floating IP address which will be mapped to its private interface using NAT. Because the management node is unaware of this mapping, we manually set the IP address of the VM host in VCL to the floating IP and modified our patch to use that value for the public IP address of all computers assigned to that host.

The end result is that each OpenStack instance is behind two layers of NAT. The first is from the floating IP address to the private IP address of the VCL management node and second is the private IP address of the management node to the private IP address of the instance.

V. Apache VCL and OpenStack Deployment

Cybera has several differences in its deployment when compared to what could be considered more standard or commonplace uses of Apache VCL.

A. *Operating System Packaging*

Currently the Apache VCL software is not officially packaged for the CentOS/RedHat operating systems, and is instead installed by downloading and extracting a compressed tar file. Cybera undertook the work to package Apache VCL into an RPM, which is the default packaging system for CentOS and RedHat. This provides several advantages.

Using an RPM package that has dependencies defined makes installation of Apache VCL quick and repeatable. Because a goal of our project was to automate the installation and configuration of Apache VCL, using packages to install the software makes the use of configuration management systems easier, especially in terms of the common package, configuration file, and service workflow.

Another useful feature of RPM packages is that the files in the package automatically have a checksum assigned. This means that in the case of a security incident the checksums of files can be easily verified using the RPM command.

B. *Automated Configuration Management*

One of the major goals for this project was to automate the installation and configuration of Apache VCL using configuration management tools such as Chef, Ansible, and Puppet. While in our project VCL is installed with an RPM package, it is not configured by that RPM.

Cybera implemented automated installation and configuration of VCL in both Ansible and Chef. In the end we found that Chef can have additional requirements, such as a central Chef server, which may add too much overhead for this particular project. Currently we deploy new Apache VCL environments using Ansible. This is not to say that Chef was unsuitable, rather that in a larger environment with more management nodes the overhead and functionality of Chef infrastructure would be more appropriate.

Using Ansible we can deploy a new Apache VCL instance, including the web front-end, the database server, and the management node, in a few minutes and do so in a repeatable, testable way. This allows us to create development and test environments relatively easily which helps to maintain a stable production server. Traditional administration usually treats servers as long-running, one-off systems. However, with cloud computing infrastructure such as OpenStack it is preferable to treat servers as systems that can be easily destroyed and subsequently recreated.

VI. Overcommitting Compute Resources

One of the key benefits of virtualization is the ability to overcommit resources such as memory, CPU, and disk. In our Apache VCL project we experimented with overcommitting. Unfortunately, the usage of Apache VCL in this project did not approach a level high enough to invoke overcommitting, but we expect that as this system moves from pilot to production the number of concurrent users will increase.

A. *Kernel Samepage Merging*

Our OpenStack compute nodes run Ubuntu Precise 12.04. By default in this version of Ubuntu a process called ksmd is enabled and running. ksmd essentially provides memory deduplication. In situations where many of the same virtual machines are instantiated, such as multiple Windows 7-based instances, ksmd can reduce memory usage by constantly scanning and deduplicating memory. This allows for more concurrent virtual machines without additional hardware.

In our current environment we have set the sleep time for ksmd to 20 milliseconds and the number of pages to scan at 20000. This is higher than the defaults of 200 milliseconds and 100 pages. Depending on those settings ksmd can use 100% of a single core, which would reduce the number of cores available to virtual machines and the hypervisor, but in our deployment this is a useful tradeoff.

B. *Solid State Drives and Windows Boot Storms*

The concept of boot storms is well known in virtual desktop infrastructure (VDI). Windows 7 virtual machines can require considerable storage input/outputs per second (IOPS) during, and shortly after, their boot process. In our experiments we found that booting many Windows 7 virtual machine in a short amount of time can require up to 8000 IOPS. Given that, depending on configuration and testing parameters, a RAID 10 array of six SATA drives can only provide approximately 400 IOPS, we had concerns regarding storage performance.

We resolved to provide more storage performance by implementing striped solid state drives in the OpenStack compute nodes. Each of our compute nodes has at least three consumer-level solid state drives installed in a RAID 0 (or striped) configuration. In testing, the slowest of these configurations provided over 50000 IOPS.

We then tested Windows 7 boot storms. In the graph “IOPS used in a windows bootstorm” (Fig. 3) the fast boot storm was 30 Windows 7 virtual machines booted 20 seconds apart. The slow boot storm, shown in red, involved the same procedure but each instance was booted 240 seconds apart.

In both cases, the system used at least 5000 IOPS. Given the results it may be that a storage system which provides a good amount of caching, such as ZFS with spinning drives and SSD caching, or other similar functioning systems, could provide enough IOPS to avoid boot storm congestion without having to resort to striped SSDs.

Interestingly, once a Windows 7 virtual machine has booted and been running for a few minutes, the IOPS usage drops considerably, to almost zero. However, during the boot process–and for a short while after booting–the virtual machines present considerable disk usage.

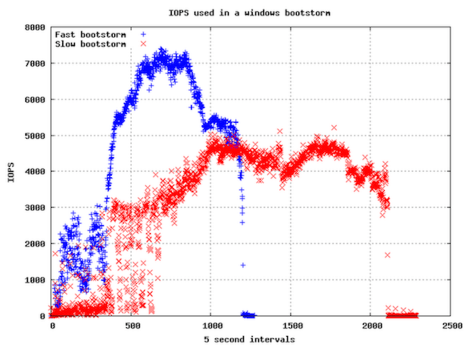

Fig. 3. IOPS Used in a Windows Bootstorm

One of the main features of Apache VCL is that when a reservation is completed the virtual machine which backed the reservation is not necessarily deleted. Instead, when a new user requests a reservation for the same environment, the same virtual machine may be re-provisioned in terms of the user data, and then re-used. This reduces the potential for boot storms, makes reservations for users faster, and reduces IOPS load as a new virtual machine does not need to be booted.

VII. Lessons Learned

During the course of our pilot project we discovered issues with our implementation, and new requirements.

A. *Golden Image Sprawl*

As our pilot project continued and new environments were created, we found that even with a relatively small number of classes the total number of golden images in our repository became unmanageable for two main reasons.

The first is that because every image is a golden image, the time it takes to update all the images with security patches grows linearly with each new image. Currently we have approximately 20 production Windows 7 golden images, which would take upwards of a week to update in terms of operating system patches.

The second issue is that because a new image can be created from any pre-existing VCL image, flaws in that image will be reproduced in the new image, and future images, in an almost viral fashion. Over time, a small mistake in the image configuration can spread across the entire image repository. For example some Windows images did not have disk checking disabled for the system disk, and at a point in time virtual machines based on these images began checking their system disk on every instantiation–a process which can take more than ten minutes. Apache VCL would then fail the reservation and computer due to a time out.

In retrospect, this golden image sprawl is a common issue with VDI implementations, and there are commercial solutions available to help resolve it. Most of these solutions enable an application virtualization or layering process in which there are perhaps only one or two golden images. Virtual machines based on those images will have various software and configurations layered on top of those golden images when they are provisioned. Instead of having 20 different golden images, there will only be one or two, and different environments are actually defined by the application virtualization or layering system.

B. *License Management*

Software licenses which had to be installed directly into the golden image complicated our workflow. More than once we had software with expired licenses and had to recreate the golden image with a new license. In almost all situations it's better to have software that requests licensing from a remote license server so that golden images do not have to be recreated. Application virtualization may also help to reduce this issue.

VIII. Future Work

A. *Multi-tenant OpenStack Cloud*

Our pilot system was deployed into a standalone OpenStack Essex cloud in which it is the only tenant. In the coming months we will deploy Apache VCL into a more modern OpenStack cloud that has multiple tenants.

B. *Non-virtualized Management Node*

Currently we have two dependencies that require us to virtualize the Apache VCL management node in the OpenStack cloud we want to use for provisioning. First is our use of OpenStack's DNS configuration to resolve computer names to IP addresses. Second is NAT which requires the management node to have access to the private network of the instances.

NAT is easily removed as a dependency if an adequate supply of public IP addresses are available; the OpenStack provisioning module would only require minor modification to automatically attach a public IP address when an instance is created. If this is not the case, a separate NAT implementation, likely integrated with the provisioning module, would need to be developed as the current patch is designed around using the management node as the gateway.

Migrating the management node out of the cloud and removing the DNS dependency would also require the provisioning module to automatically attach floating IP addresses to provide public access to the instances. Apache VCL would also need to be made aware of these floating IP addresses. If a static pool of addresses is available this could be done using current mechanisms by assigning each computer in Apache VCL one of the floating addresses as its public IP which the provisioning module could use to always attach the same address to the corresponding instance. If this isn't the case a step would likely have to be added in Apache VCL to query the appropriate provisioning module for a computer's public IP address.

C. *Gridcentric Virtual Memory Streaming Integration*

Gridcentric [6] provides an innovative product called VMS, or Virtual Memory Streaming, which integrates with OpenStack and supports the KVM and Xen hypervisors. VMS allows the launch of live images which greatly reduces storage IOPS requirements, and also provides the ability to increase the density of virtual machines on a compute node by a factor of two to five times.

Gridcentric provided patches to the Perl OpenStack API to support VMS, which will allow us to integrate it into our OpenStack Apache VCL module.

D. *Improved Remote Access*

Our pilot consisted of only Windows 7 virtual desktops and remote access was provided via Microsoft's remote desktop protocol (RDP) using various end clients. Noting that other VDI deployments often utilize different software and protocols (such as PC-over-IP or Spice) to provide remote access, we will be investigating implementing other connection methods in hopes of improving the performance of remote access, especially given students often access the virtual computer lab from home, or over campus wireless.

E. *Application Virtualization*

Based on our issues with golden image sprawl, we will be investigating application virtualization or layering technologies.

IX. Conclusion

We have found that using Apache VCL and OpenStack together is a suitable solution and meets many of our requirements for a VDI-lite system to provide a virtual computing lab. Improvements have been made to the OpenStack module for Apache VCL and the NAT patch has been contributed upstream. Both the NAT and OpenStack module are scheduled to be integrated into Apache VCL and should be part of the next release, depending on core developer workload.

During the pilot we also discovered new requirements, such as the need for application virtualization or layering to reduce the number of golden images and help with relicensing applications which cannot use a licensing server.

Through this pilot it has become apparent that there are few cost-effective solutions to our "VDI-lite" virtual computing lab needs. However, with appropriate modifications and additions, together Apache VCL and OpenStack can meet those requirements.

Acknowledgments

Cybera would like to thank the Government of Alberta, Ministry of Innovation and Advanced Education for funding this project, as well as the University of Alberta for collaboration and support throughout the project.

References

[1] https://vcl.apache.org/

[2] http://openstack.org

[3] Young, Oh, https://issues.apache.org/jira/browse/VCL-590, 12 7 2012.

[4] http://search.cpan.org/~ironcamel/Net-OpenStack-Compute-1.1002/lib/Net/OpenStack/Compute.pm

[5] Young, Oh, http://markmail.org/message/dej4uivqxkpf25ik, 17 5 2012

[6] http://gridcentric.com