Submitted by
Vidhi Jain (2014A7TS0113P)
Supriya Aggarwal (2014A7PS0013P)
Abhishek Gupta (2014A7PS0026P)

Dynamic Virtual Machine Migration by Hotspot Detection and Profiling

We report the different approaches and techniques used for this task

1. Xen hypervisor with Ubuntu
2. Xen hypervisor with openSuse
3. KVM using libvirt
4. Collectd monitoring tool
5. Hotspot detection using psutil library in Python

In the following sections, we discuss the progress made and the challenges faced.

Xen Hypervisor with Ubuntu

We installed the Xen hypervisor on the existing Ubuntu installation.

Challenges

1. There was not enough support for Xen hypervisor on Ubuntu. We also encountered dependencies issues. These issues are known and discussed but there is no feasible solution known to exist.

Xen Hypervisor with openSuse

We partitioned the physical machine and installed OpenSuse operating system.

Challenges

1. The physical machine resources were constrained (8GB RAM, 500GB harddisk) with two operating systems already installed.

KVM with libvirt

We installed qemu-kvm and virt-manager on 3 physical machines. On each physical machine, VMs were installed.
The following Virsh commands were explored: **nodecpustats, nodememstats, domstats, cpu-stats, migrate, create, list, capabilities**

NFS was set up for storing the disk images of all the virtual machines so that VMs can be migrated from one physical machine to another.
VM migration through command line and python script are working. A python script was written to collect resource usage (CPU and memory) of each VM and physical machine by using libvirt API.

Challenges:

1. We didn't know how to convert the vCPU usage reported by the libvirt API to the virtual machine utilization of the host machine. The virsh command for vCPU stats: ***getCPUStat()*** gave the cpu_time, system_time and user_time separately as well as aggregate. We attempted to find the CPU Utilization by recording the CPUStats firstly and then after an interval ***I***, we again extracted the CPUStats. We applied the following formula to get an estimate of the CPU Utilization:
   (new_cpu_time-previous_cpu_time (in nano seconds))/(Interval time in seconds*10^9)*100

Resources:

1. https://libvirt.org/docs/libvirt-appdev-guide-python/en-US/html/
2. https://github.com/abhishekgupta-1/vmMigration

Collectd monitoring tool

We used an open source tool Collectd along with Graphite web server, Carbon-cache using Whisper database.
Collectd runs a daemon process and supports a high number of plugins which allows to collect system statistics into a data store. We configured the collectd to store CPU, load and disk usage and store them in RRD files (Round robin database).
Graphite provides a monitoring platform of stats consisting of the following components:

- *Graphite-Web, a Django-based web application that renders graphs and dashboards*
- *The Carbon metric processing daemons*
- The Whisper time-series database library

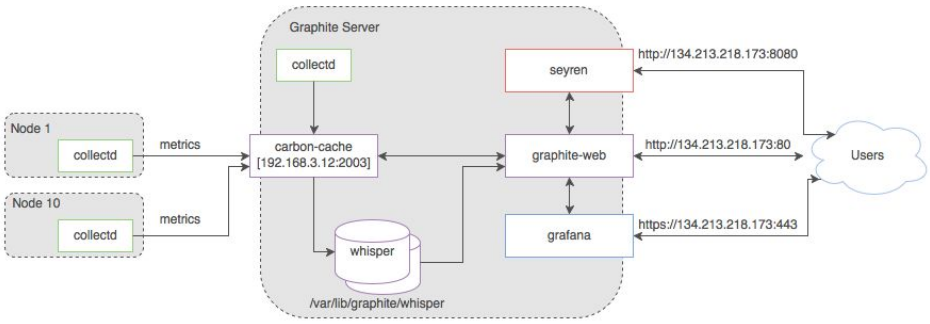

Figure 1. Architecture of graphite [Reference #1]

Collectd is set up on three machines and graphite set up was done on a single machine (controller node). Collectd was configured on the three machines to periodically send the monitored data to carbon-cache server (part of Graphite installation). Graphite is configured to then periodically fetch data from carbon-cache into the whisper database. The data was then used to generate profiles of node usage on the graphite front-end.
The system to use Collectd to gather system metrics for Graphite can support thousands of systems with dozens of virtual machines. This was perhaps an overkill for a small scale project but gave us exposure to real-world technologies used for scalable system monitoring. This provides a grey box solution but can be used as black box solution by using the libvirt plugin.

Challenges:

Configuring collectd required understanding of how Apache web Server is configured. The file collectd.conf supports a huge variety of plugins to be enabled and configured. Tweaking this required time.

The graphs generated on Graphite front end were recorded based upon the specified interval. In order to add more flexibility, we needed to consider solutions provided such as Grafana and Seyren.

References:

1. https://community.rackspace.com/products/f/25/t/6800

Hotspot detection using psutil library in Python

We discuss the system design using this library. We have observed that it retrieves the CPU percent utilization, memory percent utilization and network utilization of the system. This helps as it can be directly used for computation of Volume and Volume to Size Ratio (VSR).

Architecture:

A python script would be running on all the nodes and periodically collect the CPU percent utilization, memory percent and network utilization and send the data to a controller node via REST API.

The controller node would be a running a webserver, and a thread for each node to handle the received data from that node. Web Server receives the request, passes the information to the appropriate thread. The thread maintains a log history of last n intervals. After receiving the latest log from web server, thread updates the log history and then check if hotspot criteria is satisfied.

If a hotspot criteria is satisfied, then handler process is notified.