Resource Monitoring and Recommendation Engine for Openstack with KVM

CS677 - Operating Systems
Project Report

Submitted by:
Vijay Pasikanti, Sneha Shankar Narayan
May 8, 2014

Contents

Chapter 1

Introduction

Cloud computing platforms are complex systems with many VMs which in turn run different applications. Since applications have varying needs, different VMs are configured differently. The goal of this project is to write a simple tool that is inspired by Amazon Trusted Advisor that is used in Amazon's EC2 cloud to continuously monitor a cloud deployment and make recommendations on how to improve security, resource usage, etc for various VMs by analyzing the hypervisor. The project involves setting up OpenStack on two nodes, setting up hypervisors and virtual machines and then writing a tool that uses the OpenStack API as well as the hypervisor API to gather various statistics that are then analyzed to provide recommendations.

In this project we focus on monitoring the statistics based on which we provide resource utilization recommendations.The hypervisor that is being used here is KVM. Alerts are provided to the user when any high utilization or low utilization thresholds are hit. Recommendations are provided to the user to move machines to different nodes if the performance of the setup as a whole improves as a result of VM migration.

Chapter 2

Openstack setup

Openstack[1] is a cloud management platform where in there is a controller node and a compute node.

- The controller node runs the Identity Service, Image Service, dashboard, and management portion of Compute. It also contains the associated API services, MySQL databases, and messaging system.
- The compute node runs the hypervisor portion of Compute, which operates tenant virtual machines. By default, Compute uses KVM as the hypervisor. Compute also provisions and operates tenant networks and implements security groups. You can run more than one compute node.

We have one controller node and one compute node and the following open stack services have been installed

- **Identity service (Keystone):** Keystone is responsible for user management and service catalog.
- **Image service (Glance):** Glance is responsible for image discovery, retrieval, and storage, storing and retrieving metadata about images and for managing a storage repository for images.
- **Compute service (Nova):** Nova is a cloud computing fabric controller. It is used to host and manage cloud computing systems. Compute interacts with the Identity Service for authentication, Image Service for images, and the Dashboard for the user and administrative interface. Access to images is limited by project and by user; quotas are limited per project (for example, the number of instances). The Compute service scales horizontally on standard hardware, and downloads images to launch instances as required.

The messaging between the various open stack services is done using RabbitMQ.

Our open stack architecture is configured the following way:

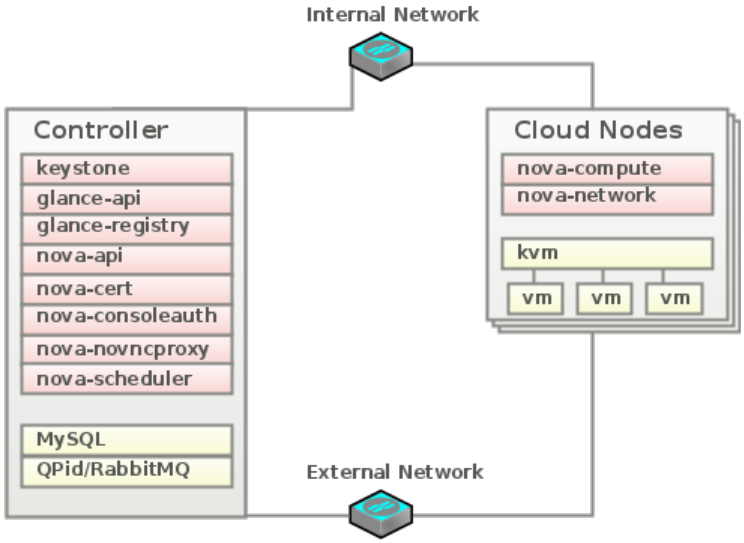

Figure 2.1: Open stack architecture

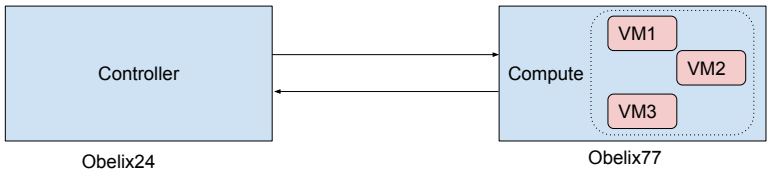

Figure 2.2: Open stack architecture

After configuring the controller and the compute node a network was setup between them. We have launched various instances of Ubuntu on our compute node. Currently we having the following VMs running in our compute node

```
root@obelix24:~# nova list
+--------------------------------------+----------+--------+------------+-------------+-----------------+
| ID                                   | Name     | Status | Task State | Power State | Networks        |
+--------------------------------------+----------+--------+------------+-------------+-----------------+
| d7cc815a-e158-4253-b4e4-65b27615fb86 | ubuntu11 | ACTIVE | None       | Running     | vmnet=10.0.0.2  |
| d53d43ba-27e0-4b45-afe0-ca566704953a | ubuntu12 | ACTIVE | None       | Running     | vmnet=10.0.0.3  |
| 0c53289c-a089-4f89-96f1-efff70fcbd9a | ubuntu13 | ACTIVE | None       | Running     | vmnet=10.0.0.4  |
| 1a1678fb-8daa-4647-89a7-414312be46d5 | ubuntu14 | ACTIVE | None       | Running     | vmnet=10.0.0.5  |
| 33251435-11f3-4013-ad30-4b4c94d99188 | ubuntu15 | ACTIVE | None       | Running     | vmnet=10.0.0.6  |
| 2b108650-c32e-4695-ae04-00622e61d8f4 | ubuntu16 | ACTIVE | None       | Running     | vmnet=10.0.0.7  |
| 56e157c6-a06f-4a6b-b152-b58664ad0c5e | ubuntu17 | ACTIVE | None       | Running     | vmnet=10.0.0.8  |
| eb54b3e2-5c5d-4c7b-a1ec-297fa96c53e9 | ubuntu18 | ACTIVE | None       | Running     | vmnet=10.0.0.9  |
| 64311806-d774-4f3a-b2bb-aacf4e6f9360 | ubuntu19 | ACTIVE | None       | Running     | vmnet=10.0.0.10 |
+--------------------------------------+----------+--------+------------+-------------+-----------------+
```

The instances are managed by the nova compute service.

Chapter 3

Existing monitoring systems explored

3.1 AWS Trusted Advisor

AWS trusted advisor [2] is a monitoring tool for Amazon Web Services (AWS), it inspects AWS environment and makes recommendations when opportunities exist to optimize costs, improve system performance, reliability and security. It provides best practices in four categories: cost optimization, security, fault tolerance and performance improvement. It covers almost all the AWS services.

Advantages:

- Provides a wide range of recommendations in almost all the AWS services.
- Developed and supported by Amazon for AWS users with new features being added continuously.

Limitations:

- Proprietary software and its premium services are charged.
- Doesn't support open source cloud platforms, for example: Openstack.

3.2 Collectd - libvirt plugin

Libvirt-plugin [6] is a part of *collectd* which uses the virtualization API libvirt. It gathers statistics about virtualized guests on a system. It collects CPU, network interface and block device usages for each guest without installing *collectd* on the guest systems.

Advantages:

- Statistics are received from the hypervisor directly, this works not only with para-virtualized hosts but with hardware virtualized machines also.
- Eliminates the need of installing a monitoring tool within each virtual machine (VM). One instance of *collectd* per hypervisor is enough to monitor all the VMs.
- Uses *libvirt* library which provides support for more then one virtualization techniques. Currently supports Xen, Qemu and KVM backends.

Limitations:

- Data collection is automatic process but monitoring and provisioning of resources is still a manual process.

3.3 Ganglia

Ganglia[7] is a scalable distributed monitoring tool for high-performance systems such as clusters and grids. It provides system level historical statistics for all machines that are being monitored. The ganglia system comprises of two unique daemons: Ganglia Monitoring Daemon (*gmond*) and Ganglia Meta Daemon (*gmetad*).

gmond: *gmond* runs on each node that is monitored. It collects system level parameters (e.g.CPU, Network Utilization) and transmits the collected statistics to *gmetad* in XDR or XML format.

gmetad: *gmetad* runs on one of the nodes of a cluster. It receives information by periodically polls a collection of data sources (defined in its configuration, usually all the nodes in the cluster it is a part of), parses the collected XML, saves all numeric, volatile metrics in round-robin databases (RRD) and exports the aggregate XML to clients. Data sources may be either gmond daemons, representing specific clusters, or other gmetad demons, representing sets of clusters.

Advantages:

- Since it is based on a hierarchical design targeted at federation of clusters, it can scale to any size of clusters or grids.
- It uses operating system calls for collecting different metrics. So, it can be used to monitor virtual machines by installing *gmond* on VMs.

Limitations:

- When used with hypervisor to monitor virtual machines, it requires *gmond* to be run on each of the virtual machines. In production deployments *admin* has no control over the applications installed on the VMs. It is hard to enforce the installation of *gmond* on each VM.

Chapter 4

Openstack monitor

The Openstack monitor that we have implemented is written completely in python, it has various modules to collect the statistics from the different virtual machines, store the statistics on disk, retrieve the stored data and provide recommendations on a web interface.

4.1 Performance metrics

The performance metrics that we have decided to use to provide resource utilization specific metrics are:

- Average CPU utilization
- Average memory utilization
- Average disk utilization
- Average network utilization
- Average disk input/output

Our aim was to analyze the above metrics and warn the user when say, the host CPU utilization becomes greater than 90 % which in turn means he needs to move the VM elsewhere or he can increase the resource allocated to the VM.

We also provide recommendations for the user to move the VMs elsewhere by creating a new compute node if we believe that the machine is overburdened. We come up with this recommendation by looking at the host level statistics. We look at the CPU and memory statistics of the host machine.

4.2 Design and Implementation

On a high level there is daemon which runs every five seconds to query the statistics from the hypervisor and this data is stored in Round Robin databases[3]. There is

another daemon which reads the data from the round robin files, provides recommendations and stores it as JSON files which is shown on the web interface which is updated every minute.

4.2.1 Collection of statistics

The various statistics that are required by our system to provide recommendations was collected using libvirt APIs. The following libvirt modules were used

- **virt-top:** virt-top [14] is similar to the unix functionality 'top' which provides data regarding the CPU, network and memory utilization of the processes in the system. Virt-top however looks at hypervisor level information and provides data that is specific to the virtual instances in the machine. The output of virt-top was streamed and statistics about the CPU, memory, disk-IO and network utilization of the VMs was collected.
- **virt-df:** Virt-df [15]is similar to the unix functionality 'df' which lists the disks that are mounted and the used and available space in a disk. Virt-df looks into the various virtual machines and reports the storage space that is in use and that is available. Thus, disk utilization was collected.
- **top:** This unix functionality is exploited to get host level statistics from the machine. This helps in providing host level recommendations to the system.

A daemon runs every five seconds and captures the data from the above modules and stores the data in Round robin database files.

4.2.2 Storing in RRD files

The main motivation behind using Round robin databases was because the RRDtool can manage time-series data effectively, won't grow infinitely in size because of the round robin nature of the database and was also easily integrable with python. RRDtool can be configured to calculate the rate of change from the previous to the current value and store this information instead.

Using the python API of RRD tool[4] the daemon that collects the various statistics also updates the RRD files associated with each of the metrics collected of each instance.

Samples of the metrics collected have been stored for the following time intervals

- 5 second averages for 1 day
- 30min averages for 5 days
- 1 hour averages for 10 days
- 2 hour averages for 20 days
- 12 hour averages for 20 days

- 12 hour averages for 20 days
- 1 week averages for 10 weeks

Also we store average maximum and minumum values for memory utilization as well for the following time intervals

- 5 second averages for 1 day
- 1 hour averages for 5 days
- 1 day averages for 10 days

4.2.3 Processing statistics and producing alerts

Another daemon has been configured to run indefinitely and it reads the statistics from the RRD files, processes them and provides recommendations. Recommendations are given based on the thresholds defined and we have defined the following thresholds.

- Average CPU utilization
  - High: 80%, to recommend user to upgrade hardware
  - Low: 20%, to alert user about resource under utilization
- Average memory utilization
  - High: 90%, to recommend user to upgrade hardware
  - Low: 20%, to alert user about resource under utilization
- Average disk utilization
  - High: 75%, to recommend user to add more disks
- Average network utilization
  - High: 10000 bytes transferred, to recommend user to increase bandwidth allocation
  - Low: 10 bytes transferred, to recommend user to downgrade bandwidth for instance
- Average disk input/output
  - High: 10000 IOps done, Alert user about high I/O
  - Low: 50 IOps done, alert user about resource under utilization
- Host level statistics
  - High CPU utilization: When a host machine is loaded to 60% CPU the user is recommended to move the VMs out.

Using the values stored in the RRD files if it is found that any of these thresholds have been hit in the past hour that fact is recorded. This daemon spits out such recommendations to JSON files which is in turn fed into the web interface when it gets updated.

4.2.4 Generating recommendations

Recommendations are generated by calculating host level resource utilization and the portion of resources each virtual machine consumed. Our work is focused on generating following recommendations divided broadly into three categories

1. Normalization of utilization on host machines.
2. Consolidation of hardware for power savings.
3. Detecting under provisioned hardware.

Normalization of resource utilization:
Over time, as virtual machines are created and deleted dynamically in a cluster of nodes providing virtual machines, there is a possibility of a scenario where a couple of nodes' resources are completely utilized by virtual machines (VM) and at the same time there are nodes with lesser utilization. In this scenario, it is advisable to move VMs from over loaded nodes to lesser ones.

For example, the following figure 4.1 shows nodes N1 and N3 are fully utilized where as N2 is partially loaded. We calculate the possibility of moving some of the VMs in N1 and N3 to N2, if N2 has enough room to accommodate, a recommendation is generated to admin to initiate a move of VMs from N1 $\rightarrow$ N2 and N3 $\rightarrow$ N2.

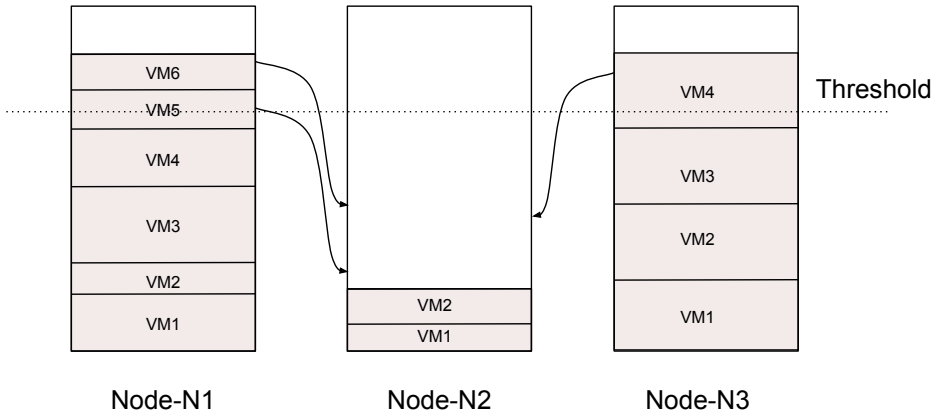

Figure 4.1: Normalization of resources

Consolidation of resources:
When there are a lot of VMs and a lot of compute nodes, there is always the possibility that at some time the nodes might be lightly loaded. In this case it is advisable to consolidate all of the VMs to fewer compute nodes and shut off all the rest to save resources.

In the following figure 4.2, we can see that the nodes N2 and N3 are lightly loaded, and thus we compute the possibility that we can move the VMs hosted on N2 and N3 to N1 and still work properly and thus we recommend to the admin that the VMs from N2 and N3 can be moved to N1.

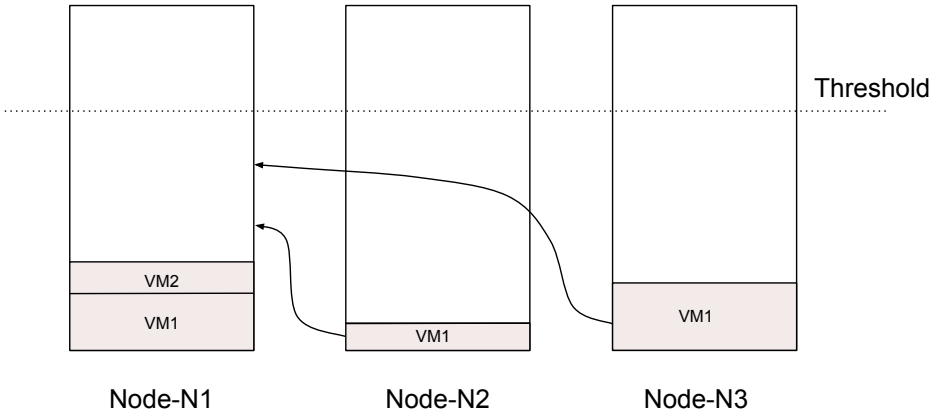

Figure 4.2: Consolidation of resources

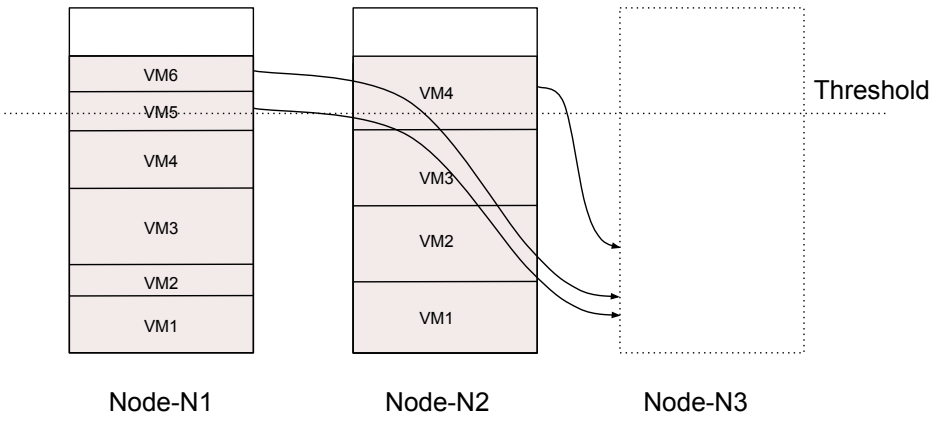

Figure 4.3: Under provisioned hardware

Under Provisioned Hardware:
Sometimes nodes that host VMs can't handle the VMs that are on them. The nodes

might be under provisioned. In such a scenario it is advisable to create new host machines and migrate the VMs to the newly created compute node. This scenario is captured in the above figure 4.3.

We can see that both the nodes N1 and N2 are over-burdened with VMs and thus we recommend that the admin move VM5, VM6 from N1 and VM4 from N2 to a new node N3.

4.2.5 Web interface

A simple python HTTP server is currently running in the compute node of the openstack setup and by using SSH tunneling we can look at the results on any browser. The webpage shows the recommendations provided by the Openstack monitor. The JSON files dumped by the daemon that processes the statistics is read using AJAX and the webpage is dynamically updated every minute with the latest data. The critical alerts are highlighted in red, low utilization and shown in blue such that a user who uses the monitoring system can take an appropriate action with respect to the configuration of the virtual machines.

Chapter 5

Testing

5.1 Setup

We used the same setup that was installed as described in the "Openstack Setup" section.
Setup details:

1. Total hosts used: 2 (1 compute node and 1 controller node)
2. Virtual machines: 9 VMs with Ubuntu 12.04 created with flavor *m1.small* (1 VCPU, 2GB memory), 1 VM with Ubuntu 12.04 created with flavor *m1.medium*

5.2 Test cases

Random loads are generated in all virtual machines using lookbusy [10], a synthetic load generator for linux systems. The following are the scenarios that we simulate:

- CPU stress: CPU is loaded from 10% to 90% randomly for upto 2 hours.
- Memory stress: The VM memory is filled upto 1.4GB randomly for upto 2 hours.
- Disk I/O stress: The VM disk is churned with file sizes upto 32MB randomly for upto 2 hours.

We randomly invoke anyone of the above scenarios continuously on all the VMs. This ensures that there is always some stressing that's happening on the machine and this is done to simulate how a VM would actually be used and this also ensures that the data we are collecting has highs and lows.

Chapter 6

Results

The recommendations/alerts are shown in the web interface as follows

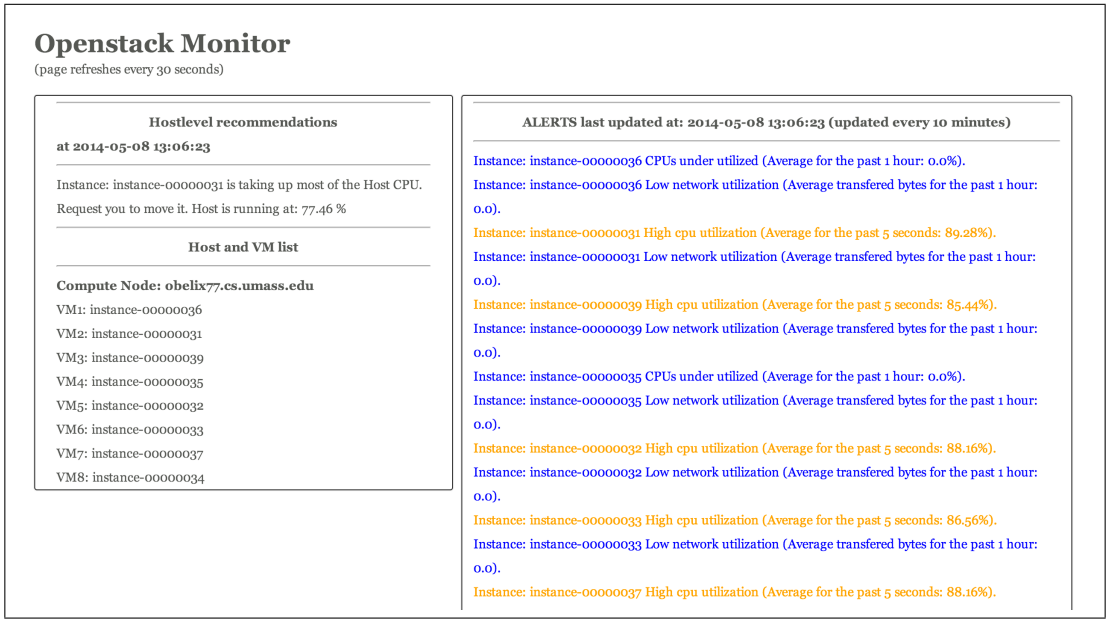

Figure 6.1: Recommendation web interface

Console output:

```
================================================
Host-level recommendations at 2014-05-05 18:52:15
================================================
Instance: instance-00000024 is taking up most of the Host CPU.
Request you to move it. Host is running at: 75.4 %
=====================================================================
ALERTS last updated at: 2014-05-05 18:52:15 (updated every 10 minutes)
=====================================================================
instance-00000024 High cpu utilization (Average for the past 1 hour: 86.12%).
instance-00000024 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
instance-00000022 High cpu utilization (Average for the past 1 hour: 89.94%).
instance-00000022 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
instance-00000021 CPUs under utilized (Average for the past 1 hour: 0.0%).
instance-00000021 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
instance-00000028 High cpu utilization (Average for the past 1 hour: 88.98%).
```

```
instance-00000028 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
instance-00000027 CPUs under utilized (Average for the past 1 hour: 0.0%).
instance-00000027 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
instance-00000023 High cpu utilization (Average for the past 1 hour: 89.88%).
instance-00000023 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
instance-00000026 High cpu utilization (Average for the past 1 hour: 90.01%).
instance-00000026 Low network utilization (Average transfered bytes for the past 1 hour: 0.0).
```

The below graphs show that the CPU utilization is constantly high for instances 22 and 24 which is why we see them in the list of alerts shown above console log.

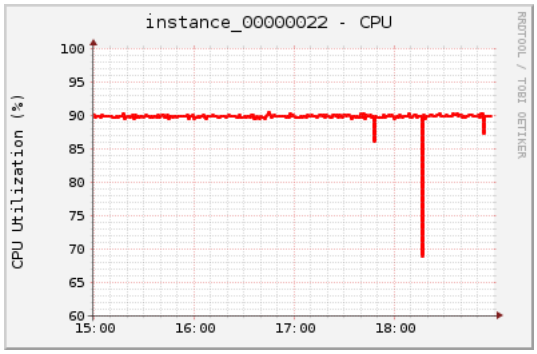

Figure 6.2: High CPU utilization on instance-00000022

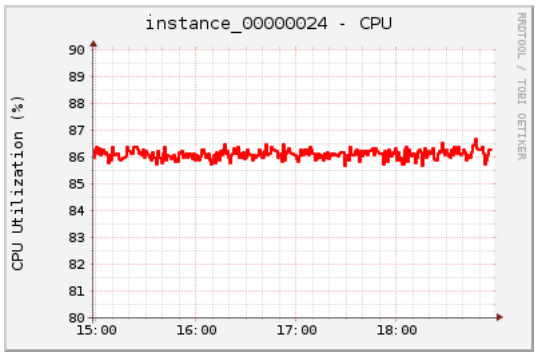

Figure 6.3: High CPU utilization on instance-00000024

The CPU utilization is almost negligible for instance 27 which is captured below.

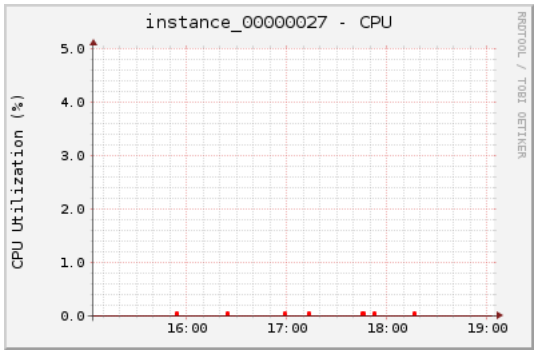

Figure 6.4: Low CPU utilization on instance-00000027

Chapter 7

Conclusion

We were able to read the various statistics like CPU utilization, memory utilization, network bandwidth utilization, and disk utilization of the virtual machines in the openstack compute node by querying the KVM hypervisor using libvirt APIs. We were able to process the data collected and provide recommendations to the user/open stack administrator to take appropriate actions regarding the virtual machine configuration. We were also able to collect statistics of the host machine (compute node) and provide the user recommendations based on that.

This monitoring system can be extended to record various other statistics of the virtual machines in order to build a full fledged open source equivalent of the Amazon Trusted Advisor.

Bibliography

[1] http://docs.openstack.org/trunk/install-guide/install/apt/content/

[2] https://aws.amazon.com/premiumsupport/trustedadvisor/

[3] http://oss.oetiker.ch/rrdtool/index.en.html

[4] http://oss.oetiker.ch/rrdtool/doc/index.en.html

[5] http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html

[6] http://collectd.org/

[7] http://ganglia.sourceforge.net/

[8] http://www.nagios.org/

[9] http://sensuapp.org/

[10] http://www.devin.com/lookbusy/

[11] http://libvirt.org/

[12] http://libvirt.org/python.html

[13] http://libvirt.org/virshcmdref.html

[14] http://people.redhat.com/~rjones/virt-top/

[15] http://libguestfs.org/virt-df.1.html