CloudView: Enabling Tenants to Monitor and Control their Cloud Instantiations

Puneet Sharma
HP Labs
Palo Alto, USA
puneet.sharma@hp.com

Sarbajit Chatterjee, Deepti Sharma
HP
Bengaluru, India
{sarbajit.chatterjee,deeptis}@hp.com

***Abstract*—The Cloud is increasingly becoming the first choice for deploying various IT services due to its flexibility and on-demand scalability. However, the cloud tenants feel handicapped by the limited visibility of the connectivity between the various compute nodes allocated when their request is instantiated by the Cloud Service Provider. In this paper, we propose, a flexible and extensible network monitoring framework called CloudView that enables tenants to monitor the connectivity between the nodes (compute and storage) of their Cloud Instantiation. We have integrated CloudView with OpenStack[4] Nova and evaluated its performance on different Cloud Service infrastructures.**

***Index Terms*—Cloud, Monitoring, Multi-tenant, OpenStack**

I. Introduction

Leveraging the Cloud for IT and other applications is a major recent trend. IDC projects that the spending on public IT cloud services will grow at 5X the rate of IT industry growth. As more and more IT services are being supported using the Cloud model, various service providers such as Amazon's EC2, Rackspace and HP's Cloud Service(HPCS) have been building multi-tenancy infrastructures. These Cloud infrastructures use virtualization and dynamic provisioning to support multiple tenants on a shared infrastructure. While this virtualization works very well for provisioning compute and storage workloads of the tenants, the network connectivity between various compute and storage (virtual) nodes has been trickier. As a result, while the provided SLAs as well as monitoring for compute and storage are well defined they are less so for the network connecting these nodes. Also, while the Cloud Service Providers allocate VMs to various tenants in accordance with the compute SLAs, the tenants do not have any control over the placement of these VMs. In the absence of network monitoring the tenants of the Cloud services have to manage their instantiations without any network knowledge. This leads to sub-optimal performance and slow reaction to failures or to poor performance events. End-to-end network measurements such as delay, loss, and available bandwidth are essential for optimizing and managing tenants' services. Different tenants might have different network monitoring requirements based on their particular services and inter-node communication. Since monitoring incurs traffic and compute overhead, the tenants might require the ability to adapt and to control the monitoring for different segments differently. To the best of our knowledge, there are no viable network monitoring frameworks that cater to the tenants' needs in Cloud service environment. In this paper we present a monitoring system that allows the cloud service users to rapidly deploy measurement and monitoring services on their compute and storage resources. Our system called – CloudView – enables the users to deploy measurement tools that are specific and relevant to their respective cloud applications. CloudView system includes a management interface for tenants to (i) perform instantaneous measurements across their nodes, (ii) schedule and manage ongoing measurements, and (iii) retrieve measurement results. CloudView manages the dissemination of measurement schedules to tenant nodes, and the retrieval of measurement results. Measurement results are stored by the system for later reference and for the purpose of validating SLA compliance. Measurement results can also be used by the tenants to optimize application placement and for migration of their VMs after the original request instantiation. We implemented CloudView with opensource OpenStack platform and evaluated it on different cloud service provider infrastructures.

The rest of this paper is organized as follows: Section II describes the design goals for tenant cloud monitoring system. Section III discusses CloudView architecture followed by OpenStack implementation details in Section IV. We present evaluation results for CloudView overhead and a sample tenant control application in Section V. Related work is presented in Section VI and the paper concludes with future research directions in Section VII.

II. Design Goals

In this section, we describe the design goals of our system – CloudView – that provides a framework for cloud tenants to make controlled active and passive measurements between (virtual) nodes after their new instance creation request has been granted by the Cloud-Service Provider (CSP). CloudView was designed with the following design goals in consideration:

- **End-point based**: The Cloud tenants do not have control and access to invoke measurements from the cloud network elements. Hence, a monitoring platform must be end-point based. Primarily these end-points are the compute VMs and storage elements allocated by the cloud provider to the tenant.
- **Extensibility**: A key feature of our design is extensibility, allowing the user to utilize arbitrary measurement tools. The measurement tools need to be adequately described. We

978-3-901882-50-0

provide a mechanism to do so. For example, to measure available bandwidth between two end-nodes, the user may choose to use existing tools such as pathChirp, pathload, Spruce; or they may develop and use a completely new tool. The users might also be interested in installing tools that measure the combined performance of network and the VMs such as httperf [5] to measure the webserver response from a particular end host agent. We designed CloudView in such a way that third-party measurement tools could be integrated with minimal modification. This is also important for tenants that want to use measurement tools they do not have ability to modify.

- **Portability**: Another key feature is portability because the resources that are required by the tenants and those that are available from the Cloud Service provider might be extremely heterogeneous. For instance, the service providers might only support a subset of operating systems. Similarly, different tenants can have restrictions on choice of OS that their applications can run on.
- **Metric Semantics**: There are various measurement tools for a single metric such as latency, loss etc. In order to make the system orthogonal to tool used for measuring a particular metric, tenant cloud monitoring system should allow associating measurement semantics with metric collection; thus allowing tenants to be agnostic to the exact tools being used and still allowing them flexibility to pick tool of their choice for a particular metric.
- **Monitoring Customization**: Tenants have varying monitoring requirements depending on how the measurement information is used. Some users are only interested in instantaneous measurements before initial deployment of their application on the cloud. While other users might want to customize the measurement frequency depending on the rate at which control actions such a VM migration and job assignment can be taken in their deployment.
- **Ease of Use**: It is important that the monitoring system is easy to use so that tenants do not incur heavy installation overheads and can integrate monitoring data with their application controllers easily.

III. Architecture & Implementation

Our design of CloudView leverages our earlier work on Scalable Sensing Service [1]. CloudView has two primary components:

***Sensor pod*:** A sensor pod is a light-weight and extensible web service enabled framework which hosts measurement sensors (e.g., ping, pathChirp, tulip etc.). Sensor pod needs to run on each node which serves as an end point (source or sink) of a measurement. Figure 1 depicts the components of the sensor pod. Sensor pod processes measurement requests via a built-in web server. The services on the sensor pod are written in Python, and contain a framework for plugging in new sensors. Sensors extend the interface provided by CloudView to manage measurement tools. Each sensor provides three functions: (i) perform the tenant requested measurement, (ii) process and return the measurement results, and (iii) clean up any measurement-related resources when done.

The sensor pod logic includes the framework for supporting

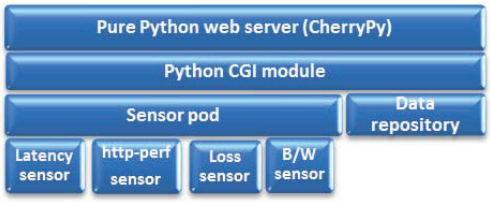

Figure 1 CloudView Sensor Pod

sensors. It provides the web service interface which is invoked by CloudView information manager. It also provides scheduling and invocation services for taking instant (non-scheduled measurements requested by the user in real-time) and periodic (repeating measurements scheduled in advance) measurements. The data repository stores the measurement data collected by sensors in periodic measurements for later retrieval.

***CloudView Information Manager*:** CloudView Information Manager is the application with which the user interaction happens. The manager triggers measurements on the sensor pods and collects data. It includes a web application (portal), written in Java/JSP/JavaScript.

CloudView uses relational database (MySQL [11]) to better organize the high volume of data that is generated from sensors running in periodic fashion. We treat all the data stored by CloudView as separate entity. Each entity forms a database table having relationship with the other entities. User details are stored in a separate table with reference to project table which in turn is connected with the node list. Sensors are stored separately from the parameters they use and metrics they generate, to make the future alterations easier. The periodic measurement data is stored in 2 linked tables. One holds the measurement details and its id. The other table stores the raw data in key value format with a reference to the measurement that generated it.

CloudView Information Manager can be installed on any host which can establish communication with the tenants' nodes hosting sensor pods. This includes installation on tenant's node or on some other host.

We now discuss two salient capabilities of the CloudView design that provide the tenant with the flexible and extensible monitoring control.

```
<?xml version="1.0" encoding="UTF-8" ?>
   <sensor xmlns="http://www.illusion.hpl.hp.com"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.illusion.hpl.hp.com
                      sensor-schema.xsd">
      <name>httperf</name>
      <description>It collects http data</description>
      <module_name>httperf.py</module_name>
      <parameters>
         <parameter>
            <name>server</name>
            <description>server url</description>
            <type>string</type>
            <unit></unit>
            <default_value>google.com</default_value>
            <max_value></max_value>
            <min_value></min_value>
            <regex></regex>
            <required>true</required>
         </parameter>
         <parameter>
            <name>port</name>
            <description>port no.</description>
            <type>integer</type>
            <unit></unit>
            <default_value>80</default_value>
            <required>true</required>
         </parameter>
      </parameters>
      <provides>
         <metric>
            <name>connection rate</name>
            <description>connection rate</description>
            <type>float</type>
            <unit>ms/conn</unit>
         </metric>
         <metric>
            <name>net io</name>
            <description>net io</description>
            <type>float</type>
            <unit>KB/s</unit>
         </metric>
      </provides>
   </sensor>
```

Figure 2 XML Description of httperf sensor

Sensor Pod Extensibility: The CloudView sensor pod provides a pluggable sensor interface for which individual sensor management plug-ins (hereafter simply called sensors) may be provided. A sensor consists of a sensor description in XML and a Python module implementing the CloudView sensor interface which performs the actual sensing (or invokes an external program to do so). This allows the tenants to add new sensors that are relevant to their instantiation. For instance, a tenant deploying a multi- tiered Web application might want to monitor the performance of its web servers using tools like httperf[5] as shown in Figure 2.

Sensor Pod Portability: Our sensor pod is implemented entirely in platform independent interpreted code. This enables a single sensor pod build to run on a variety of platforms and greatly simplifies deployment. The basic functionality of sensor pod is provided in Python, reducing its platform requirements to a working Python 2.5 interpreter and a Bourne-compatible shell (for installation and startup). The sensor pod uses the CherryPy (http://www.cherrypy.org/) web server engine for interfacing with the information manager. Hence, the Python code dependencies are reduced to pure Python code, with any optional and/or unnecessary native code removed, and these pure Python support libraries are bundled with the sensor pod software.

IV. OpenStack Integration

OpenStack[4] is one of the most commonly used opensource Cloud Service framework. It is used by various service providers such as HPCS and Rackspace. OpenStack [4] provides compute (Nova), object storage (Swift), image service (Glance) and authentication service (Keystone). There are other services (e.g. networking service [Quantum]) being developed based on core OpenStack [4] APIs. OpenStack allows developers to access cloud features over easy to use APIs. Similar APIs and information is provided by the platforms used by other Cloud Service Providers such as AWS that do not use OpenStack [4]. In order to validate the feasibility and performance of the CloudView architecture we integrated CloudView with OpenStack-Nova as shown in Figure 3. CloudView uses Django [1] framework's Django-Nova python API to interact with Nova. Django-Nova provides web based interactions with the OpenStack Nova cloud controller (Nova API). CloudView is integrated with OpenStack REST based API as well. We also implemented web-app based integration of CloudView with OpenStack dashboard to provide configuration and instantiation information. OpenStack supports Keystone authentication service to provide single-sign-on for all OpenStack services including compute. CloudView leverages Keystone authentication services for accessing the cloud instantiations of the users.

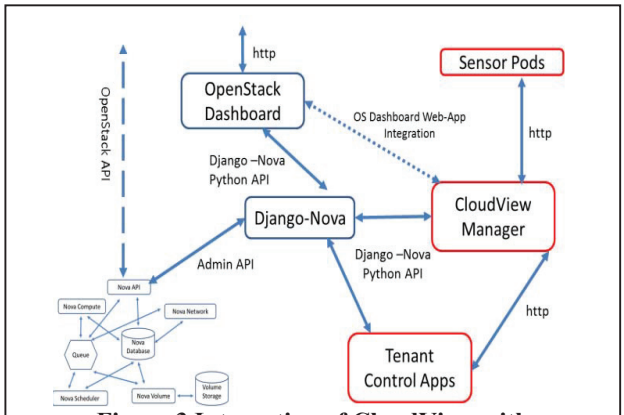

Figure 3 Integration of CloudView with OpenStack based Cloud Ecosystem

Once user accounts have been created in the CloudView Information Manager, users can create new projects for different cloud instantiations using the dashboard or CloudView command-line API. The user needs to provide access key, secret key, tenant id, Keystone authentication URL and the SSH credentials for creating new projects. This information is available from cloud service provider. Using the

project information that the user has provided, the list of instances is fetched from OpenStack compute service. The user can further select subset of VMs for inclusion in the project and CloudView will initiate automated sensor pod installation on those VMs.

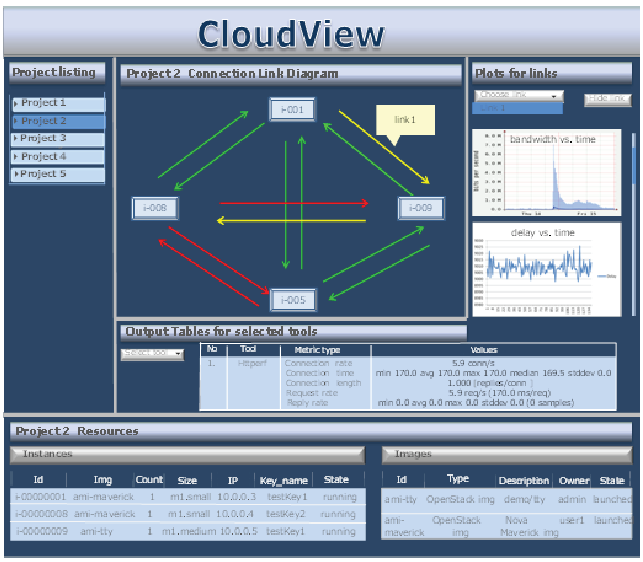

Figure 4 CloudView's Information Manager Dashboard Integrated with OpenStack

Similarly, tenants can add new sensors by providing a XML description of the new measurement tools and the python module for tool invocation and result processing. The new sensors are installed and activated on all the VM nodes of the project. CloudView Information Manager also provides interface for configuring and initiating long running periodic measurements when a new project or sensor is added. User can also perform an instant measurement by specifying the sensor and the requisite parameters for one time measurement.

Figure 4 shows the snapshot for CloudView's Dashboard showing the OpenStack integration. This snapshot shows how a tenant has configured CloudView based dashboard to monitor the network properties such as delay and available bandwidth between VMs for different projects. Using the extensible sensor-pod, we also added tools like httperf [5] for measuring the performance of web-service. All it required was describing the sensor schema for httperf [5] in XML.

V. Evaluation

In this section we present the results from evaluation and testing of CloudView that we did on different cloud infrastructures.

A. Portability

We were able to deploy CloudView on Cloud Services provided by 2 different vendors, validating its easy portability. The results presented in this paper are from three separate cloud instantiations. These instantiations are referred to as CSP1-zone1, CSP1-zone2, and CSP2 in this paper. CSP-zone1 and CSP-zone2 instantiations were done on different administrative zones of the same cloud service provider.

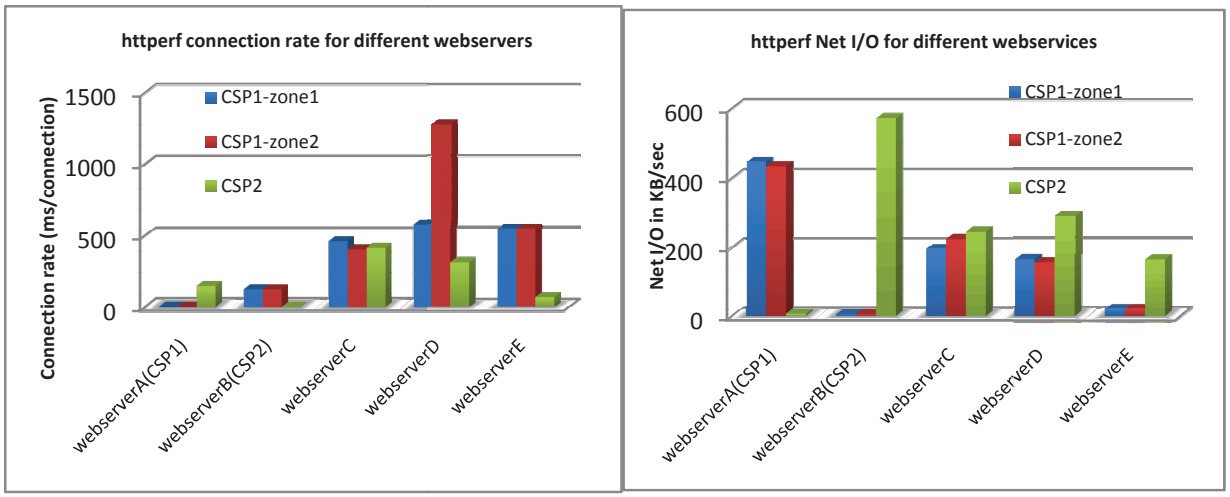

Figure 5 Webservice Performance from different Cloud Service Instantiations

CloudView was also successfully instantiated on a variety of VM OS and configurations such as Ubuntu, Redhat, CentOS. Given that sensor pod implementation is in Python, it can be ported to any OS with support for Python runtime.

Table 1 Avalaible BW(Mbps) between CSP Instances

	CSP1-zone1	CSP1-zone2	CSP2
CSP1-zone1	15	14.33	13.67
CSP1-zone2	15.33	17	14.67
CSP2	10.67	10.33	22

Table 2 RTT (ms) between CSP Instances

	CSP-zone1	CSP-zone2	CSP2
CSP1-zone1	0.633	0.682	61.952
CSP1-zone2	0.683	0.641	62.202
CSP2	62.201	62.191	0.567

Periodic latency and available bandwidth measurements were performed between the three cloud instantiations. These metrics were measured using *abget* [13] and *ping*. Tables 1 and 2 show the intra- and inter- instance available bandwidth and average round trip time (RTT) over the measurement period. It can be observed that RTT or latency between the VMs instantiated at different CSPs is considerably higher than intra-CSP RTT or latencies. At the same time, the available bandwidth for inter-CSP VMs is lower than intra-CSP VMs. Though this is expected, monitoring systems like CloudView allow tenants to customize and precisely measure these properties and use them for improving performance of their cloud offerings.

B. Extensibility

We also explored the extensibility of the CloudView system by dynamically adding new measurement tools. Figure 2 shows an example for the xml file used for adding httperf tool for measuring the webserver performance. This description includes sensor input parameters and alsocollected metrics. The same sensor description is used by the information manager to interpret the measurements. Figure 4 also shows an instant httperf [5] measurement performed using CloudView.

We used CloudView to perform httperf measurements to study the performance of different public and private webservices when accessed from VMs hosted in different cloud service providers. Figure 5 shows two metrics for webservice performance collected using httperf invocations. We selected 5 webservers and measured their performance from the cloud instantiations on three different locations, namely, CSP1-zone1, CSP1-zone2 and CSP2. Out of the 5 webservers, three were well-known websites (labeled here as webserverC, webserverD and webserverE) and we also hosted two webservers on our cloud instantiations at the two providers {labeled here as webserverA (CSP1) and webserverB (CSP2)}. We configured 15 pairs of periodic httperf measurements between the 3 instantiated VMs and 5 selected webservers and collected two metrics from these measurements: i) connection rate and ii) net I/O. The Net I/O metric reported by httperf represents the network throughput between the server and the client. The connection rate is the time taken by the server to initiate the http connection. It can be observed that both connection rate as well as Net I/O varies greatly depending on the webserver as well as the location of the client VM in the cloud. As expected, we also noticed variation in the performance based on the specifications of the cloud VM. Such measurements can be used by tenants to appropriately place their webservers in cloud infrastructure to meet the SLAs of the clients accessing their service. Similarly, tenants can optimize their cloud instantiations by requesting suitable VMs from CSPs based on measurements from CloudView.

C. Sensor Pod Overhead

We profiled the CPU and memory overhead of running the sensor pods on some representative VMs available in CSP1 and CSP2 infrastructure. The CSP1 VM was 2 vCPU with 623MB of RAM, whereas CSP2 VM was 1 vCPU with 1GB of RAM. Both the VMs were configured to run Ubuntu 64bit OS. On each of the VMs we incrementally ran CloudView sensors and measured the CPU and memory overhead increase. It was observed that the sensors (i.e. the measurement tools) are the major contributors to the monitoring resource usage. The sensor pod itself introduces close to 0.5% CPU and memory overhead beyond the raw invocation of the sensors.

VI. TENANT USE-CASES

This section presents two representative tenant use cases for tenant controlled cloud monitoring service like CloudView. One of the applications is purely for monitoring the cloud instantiation, while the other one uses measurements for controlling job placement. In order to study the utility of CloudView, we instantiated a few representative cloud applications such as Hadoop clusters and 3-tier webservices on VMs in two different provider infrastructures.

A. Cloud Instantiation Dashboard

As a simple tenant cloud monitoring application we implemented a CloudView Dashboard for visualizing the monitored information such as path properties between different VMs. The dashboard also allows customized monitoring of tenants' cloud service instantiation using application monitoring tools such as httperf. Figure 4 shows a screenshot of our dashboard. Besides visualization and querying the measurement results collected by CloudView, the dashboard included color-coded visualization of the health of virtual connectivity graph between the VMs of particular cloud instantiation. Periodic measurements were invoked to collect current path properties, such as delay, loss and available bandwidth between the VMs. A threshold based color-coded health indication is displayed on the dashboard as the path properties change. This kind of dashboard can be used by the cloud application administrators to detect faults and performance degradation in their cloud instantiations and trigger corrective action.

B. Hadoop Job Placement Control

Hadoop is one of the most common open-source frameworks for data-intensive distributed computation that is used extensively by cloud application developers, including Yahoo and Facebook. Users like these manage and operate their own cloud datacenters and hence can control the VM allocation and placement for their hadoop jobs. However, users that use other service providers have limited control over where their VM instances are placed. Hence the only control knob such users have is the placement of hadoop jobs. In this use-case for CloudView, we demonstrated how tenants can collect connectivity information between their VMs and use this information to improve the job completion times.

We have used CloudBurst [10] application for measuring hadoop job performance. Cloudburst [10] is developed for use in various biological analyses like SNP (single-nucleotide polymorphism) discovery, genotyping, and personal genomics etc. It is basically a read-mapping algorithm for mapping next-generation sequence data to reference genomes (e.g. human genome). Cloudburst is optimized to take advantage of parallel processing capabilities of hadoop. For our study we used two Jobs:

[Job1]: Using the reference genome sequence of Streptococcus suis bacteria as the original data set we created a small job with two sub-tasks. Each sub-task had 100K illumina reads [12] as the query and there was no communication required between the two sub-tasks.

[Job2]: Using the reference genome sequence of Gorilla as the original data set we created a large job with two homogenous sub-tasks. Similar to Job1, each sub-task of Job2 also had 100K illumina reads as the query and there was no communication required between the two sub-tasks.

These jobs were executed on a cloud instantiation with 10 VMs. We emulated two clusters of 5VMs each by artificially inserting delay of 100ms between the VMs belonging to different clusters. These clusters might represent VMs in different zones of same service provider or different service providers. For each job execution 8 VMs were allocated to the hadoop cluster. Two strategies were used for placing the sub-tasks on the 8 pre-allocated VMs. First placement strategy, called *random-placement* allocates VMs to the two sub-tasks randomly. The other placement strategy uses link characteristics measured using CloudView to create VM clusters that are close to each other. Then each of the sub-tasks is assigned to the VMs belonging to the same cluster. We called this placement *connectivity aware placement.*

We monitored the hadoop job tracker interface for the job execution information and measured the job completion time. Figure 6 compares the job completion time for the two job placement strategies. We noticed only 1% improvement in completion time for Job1 with connectivity aware placement strategy. In the case of the larger job (Job2), the measurements collected using CloudView helped the connectivity aware placement finish faster than random placement by over 20%.

VII. RELATED WORK

Most of the earlier work on monitoring cloud application has focused on network operations and characterizing the traffic characteristics of the cloud applications [14]. While these tools and studies are important for the infrastructure providers, they do not allow tenants to customize the measurements to their requirements.

Workload and network topology information has been used for resource allocation in earlier work [15, 16]. Our work is orthogonal to this earlier research as it allows the users to collect and monitor network topology and performance information for job placement. Also, earlier research assumes that the job assignment is being done in a private cloud environment where the users can control the placement of their VMs based on their workload. However, such VM placement control for users is restricted in public cloud infrastructures.

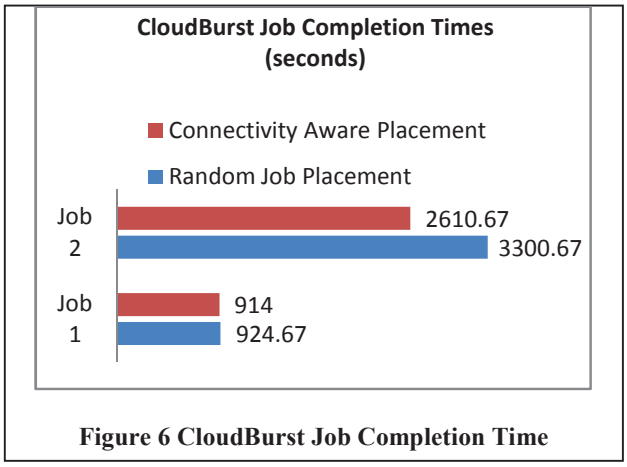

Figure 6 CloudBurst Job Completion Time

There are some tools for monitoring cloud instances and network as a whole. Icinga [6] is one such example. Icinga is an open source network monitoring system which can be deployed in a large scale network. It is built on Nagios [7] core. Icinga provides custom sensor/plugin addition and push based data fetching similar to CloudView. But it was not developed with cloud environment (especially OpenStack) in mind. Also Icinga plugins are not platform independent which is not the case with CloudView sensors.

Amazon CloudWatch [8] is well-equipped for cloud environment. It is integrated with AWS web console and monitors AWS instances. CloudWatch allows graph visualizations and programmatic access to the monitoring data like CloudView. But custom metric addition is difficult in CloudWatch and it can only be used in AWS environment. So, if we have a pan CSP cluster, CloudWatch may not become the default choice. CloudView can be adapted to any environment because of its platform-independent nature.

VIII. CONCLUSION AND FUTURE WORK

In this paper we presented the design and implementation of CloudView, a customizable monitoring system for tenants' cloud service instantiations. The extensibility and portability of CloudView was demonstrated by adding new application monitoring tools such as httperf and abget and deployment testing on different cloud service infrastructures. The utility of tenant monitoring of cloud instances to control their applications was evaluated using CloudBurst application.

Extensions of this work include feeding monitoring information to tenant cloud service controller applications. CloudView measurements can be integrated with tenant cloud control applications such as workload migration, failure driven migration etc. In future, CloudView should also include node specific information from Cloud monitoring tools and APIs used by the administrators.

REFERENCES

[1] Django: https://www.djangoproject.com/

[2] Ethan Blanton, Sarbajit Chatterjee, Sriharsha Ganga, Sumit Kala, Deepti Sharma, Sonia Fahmy, Puneet Sharma, "Design and Evaluation of the S3 Monitor Network Measurement Service on GENI," Proceedings of the Fourth International Conference on COMmunication Systems and NETworkS (COMSNETS), Bengaluru (Bangalore), India, January 2012.

[3] Praveen Yalagandula, Puneet Sharma, Sujata Banerjee, Sung-Ju.Lee, and Sujoy Basu, "S^3: A Scalable Sensing Service for Monitoring Large Networked Systems," Proceedings of ACM INM 2006(in conjunction with Sigcomm 2006), Pisa, Italy, September 2006.

[4] OpenStack: http://www.openstack.org/

[5] httperf: http://www.hpl.hp.com/research/linux/httperf/

[6] ICINGA-open source monitoring: https://wiki.icinga.org/

[7] ICINGA –Nagios: https://www.icinga.org/nagios/

[8] Amazon CloudWatch: http://aws.amazon.com/cloudwatch/

[9] D. Antoniades, M. Athanatos, A. Papadogiannakis, E.P. Markatos, and C. Dovrolis, "Available bandwidth measurement as simple as running wget," In Proceedings of the Passive and Active Measurement Conference (PAM2006), March 2006. http://www.ics.forth.gr/~papadog/abget/

[10] CloudBurst: http://sourceforge.net/apps/mediawiki/cloudburst-bio/index.php?title=CloudBurst

[11] MYSQL relational database: http://dev.mysql.com/doc/refman/5.0/en/what-is-mysql.html

[12] Reference genome and 100K illumine sequence data available from:
http://www.sanger.ac.uk/resources/downloads/

[13] Abget b/w measurement tool: http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/bw-est/abget_tutorial.html

[14] Theophilus Benson, Aditya Akella and Dave Maltz, "Network Traffic Characteristics of Data Centers in the Wild," IMC 2010, Melbourne, Australia.

[15] Gunho Lee, Niraj Tolia, Parthasarathy Ranganathan, and Randy H. Katz, "Topology-Aware Resource Allocation for Data-Intensive Workloads," Computer Communications Review, January 2011.

[16] B. Viswanathan, A. Verma, S. Dutta, "CloudMap: Workload-aware placement in private heterogeneous clouds," IEEE Network Operations and Management Symposium (NOMS), 2012.