Resource Usage Monitoring in Clouds

Mohit Dhingra, J. Lakshmi, S. K. Nandy
CAD Lab, Indian Institute of Science
Bangalore 560 012 (India)
Email: mohit@cadl.iisc.ernet.in, jlakshmi@serc.iisc.in, nandy@serc.iisc.in

***Abstract*—Monitoring of infrastructural resources in clouds plays a crucial role in providing application guarantees like performance, availability, and security. Monitoring is crucial from two perspectives - the cloud-user and the service provider. The cloud user's interest is in doing an analysis to arrive at appropriate Service-level agreement (SLA) demands and the cloud provider's interest is to assess if the demand can be met. To support this, a monitoring framework is necessary particularly since cloud hosts are subject to varying load conditions. To illustrate the importance of such a framework, we choose the example of performance being the Quality of Service (QoS) requirement and show how inappropriate provisioning of resources may lead to unexpected performance bottlenecks. We evaluate existing monitoring frameworks to bring out the motivation for building much more powerful monitoring frameworks. We then propose a distributed monitoring framework, which enables fine grained monitoring for applications and demonstrate with a prototype system implementation for typical use cases.**

***Index Terms*—Clouds, Monitoring, Quality of service, Performance analysis, Virtual machine monitors.**

I. Introduction

Cloud computing enables provisioning of a software, platform or infrastructure as a utility to users. The underlying technology that allows sharing of servers' infrastructural resources like processing cores, memory and I/O devices is virtualization. However, varying infrastructural and service loads may significantly impact the performance of an application running on cloud hosts. As a result, building framework that enables Service Level Agreements (SLAs) based on an application's QoS requirements [1] like performance guarantees, security levels, reliability and availability constraints plays an important role for cloud adoption. Monitoring of infrastructural resources is essentially the first step for building such frameworks.

Monitoring can be done for various service models of the Cloud. Service models like Platform as a Service (PaaS) and Software as a Service (SaaS) are a result of the abstractions built over the Infrastructure as a Service (IaaS) model. In order to monitor at the application or platform level, it becomes mandatory to have necessary monitors in place for the infrastructural resources. Unless performance guarantees at the level of hardware resources like CPU, Memory and I/O Devices are not given, there is no way that an application's performance can be guaranteed [2]. In other words, PaaS and SaaS models cannot guarantee performance unless a monitoring and control framework for IaaS model exists. Hence, as a first step, we explore the resource monitoring frameworks for IaaS clouds.

Both Cloud provider and clients (which could be Service providers in case of PaaS Clouds, or end users) are the beneficiaries of resource monitoring. Cloud providers have to monitor the current status of allocated resources in order to handle future requests from their users efficiently and to keep an eye on malicious users by identifying anomalous usage behaviour [3]. Monitoring is also beneficial to the end-users since it helps them to analyze their resource requirements, and ensure that they they get the requested amount of resources they are paying for. Also, it enables them to know when to request for more resources, when to relinquish any underutilized resources, and what proportion of various physical resources are appropriate for the kind of applications they are running.

The rest of the paper is organized as follows: Section II provides a few experimental results which motivate the need for a strong monitoring framework from the cloud-user's perspective; Section III provides analysis on the capabilities and limitations of a few existing cloud monitoring frameworks; Section IV provides proposal for a distributed resource monitoring framework which attempts to overcome the limitations discussed; and Section V concludes the discussion.

II. Motivation for Monitoring

Consider an example of a web server. The usage pattern of a web server depends on various factors. One such factor is the time of the day. For example, a server hosting a banking website is likely to have more hits during day-time when most of the transactions take place instantaneously rather than at night-time. Similarly, a web server hosting news is likely to have more hits on the occurrence of some unusual event like a Tsunami. Web servers need to maintain sufficient resources so as to provide uninterrupted service to end users even during peak usage. However, this approach could keep the associated resources under utilized mostly [4]. An alternative approach to handling such scenarios is to map the web servers to a Cloud Infrastructure, which would take care of the elastic requirements of a web server and also result in an economically viable model. Consequently, in the following sections, we analyse the web server hosted on Cloud. For this analysis, we use httperf [5] tool as representative of the web server workload.

A. httperf : A case study

Httperf is a benchmarking tool used to measure web server performance. It runs on client machines to generate a specific HTTP workload in the form of number of requests per second.

By varying the characteristics of the generated workload, we analyse usage patterns of physical resources, maximum achievable throughput, and the response time [6]. Physical resource usage patterns are observed to identify resources that act as bottlenecks leading to system saturation. Throughput and Response time helps determine the request rate when the system gets saturated. The goal of this experiment is to understand the different resources contributing to the performance of the application.

B. Experimental Setup

Table I lists the characteristics of the computing resources we used during our experiment. OpenNebula cloud computing toolkit [7] is used to build IaaS cloud with Xen [8] as Virtual Machine Monitor (VMM) or hypervisor. Xen boots into a privileged hypervisor, called Dom0, with exclusive access to the hardware. In our setup, Dom0 is OpenSUSE 11.4 with Xen aware kernel.

TABLE I
MACHINE SPECIFICATION

HW-SW	Physical Machine	Virtual Machine
Processor	Intel i7 Quad-Core 3.07 GHz	Intel i7 One Core 3.07 GHz
Memory	8 GB	1 GB
Storage	512 GB	8 GB
Platform	OpenSUSE 11.4 Xen Kernel	OpenSUSE 11.4
Network Bandwidth	1 Gbps[1]	N.A.[2]

[1] Same Network Interface Card is shared by all VMs using Xen paravirtualized driver.
[2] Virtual Machines are connected through software bridge, without any control/limit.

The table also lists the specifications of a Virtual Machine (VM) created by OpenNebula toolkit, which is configured to use Xen driver for Virtual Machine Management and Information Management. All virtual machines created in our experiments are identical in terms of their specifications. Figure 1 shows the different components of the experimental setup. Three Web Servers are hosted on three virtual machines on a single host, with httpd as the program serving the http requests. Client1, Client2, and Client3 simultaneously run httperf benchmark tests for VM1, VM2, and VM3 respectively. Since Dom0 has elevated privileges, it is displayed along with the hypervisor.

C. Experimental Results

Figure 2 shows the experimental results. Figure 2(a) shows the variation of Net I/O throughput with varying http request rates from the client running httperf tests. Net I/O data rate measures the actual network data rate on TCP connections, excluding headers and retransmissions. Figure 2(b) shows the variation of response time with varying http request rates. Response time captures the average time the server takes, to respond to requests. Both figures show that all VMs get

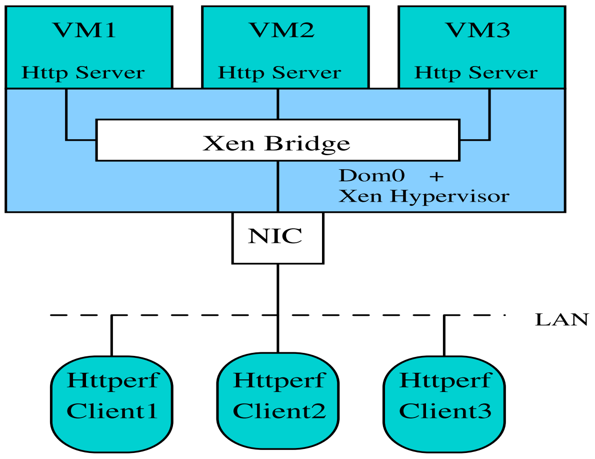


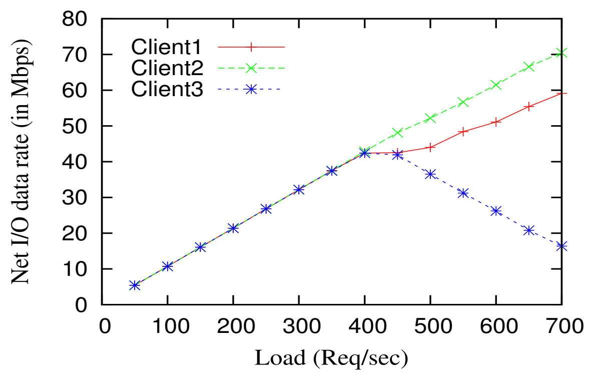
Fig. 1. Experimental Setup

saturated at 400 requests per second, as response time increases sharply and Net I/O shows random distribution among VMs, beyond this request rate. After VMs are saturated, timeout errors also increase sharply as web servers are unable to handle requests exceeding their saturation limits, so they start dropping packets leading to timeouts and subsequent retransmissions. Both throughput and response time metrics are measured at the client side.

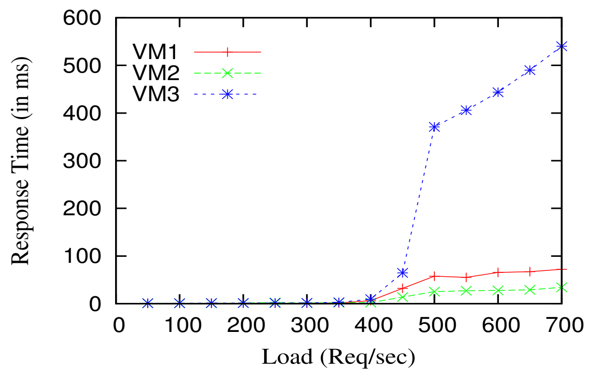
To understand the resources contributing to the performance of the httperf client, we observe the resource usage on the cloud host. We notice that both the VM and Dom0 CPU usage, and network bandwidth contribute to the behaviour of httperf client. This is due to Xen virtualization architecture used. In order to measure resource usage at server side, we used the XenMon [9] tool to measure CPU usage for Dom0 and guest VMs (called DomUs). For the experiment, each of the DomUs and Dom0 are pinned to a particular CPU Core. Figure 2(c) shows the CPU usage with varying http request rates. The output shows that there is a drastic difference between the CPU usage of Dom0 and DomUs. Dom0 shows more than 90% CPU usage when the system gets saturated. This suggests a strong possibility of Dom0 CPU being a performance bottleneck leading to system saturation. On the other hand, all VMs consume just under 20% CPU even at the time of saturation. Section II-D describes the reason for such an unexpected behaviour of the system.

D. Analysis

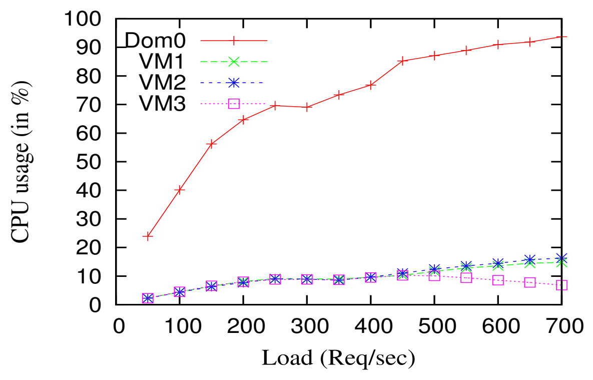
In our setup, Dom0 hosts the physical device driver for the network interface. To support network device virtualization, Dom0 hosts paravirtualized backend driver over the physical device driver and all other guest VMs host the corresponding frontend driver. All of the incoming packets are first processed by Dom0's backend driver, where it identifies their destination. Dom0's backend driver can either copy the packet buffer from its address space to the guest VM's address space, or it can use the zero-copy page-flipping technique. Considering the network packet size, copying data is faster than flipping the



(a) Net I/O throughput with varying httperf load



(b) Response Time with varying httperf load



(c) CPU Usage with varying httperf load

Fig. 2. Experimental Results

pages. This entire process involves high CPU usage by Dom0. Because high usage of CPU by Dom0 involves processing of packets for other domains, there arises a need for handling CPU division between domains carefully and appropriately. This gives us the motivation for a monitoring framework that gives a fine grained view of resource usage, at least in Dom0, so that the cloud user knows what resources to ask for and the cloud provider knows how to efficiently distribute the resources. Also, fine-grained monitoring of resources can lead to fairer resource accounting schemes.

TABLE II
MONITOR INFORMATION FROM OPENNEBULA MONITORING FRAMEWORK

Metrics	Value
CPU	15 %
Memory	1048576 Bytes
Net TX	133181200 Bytes
Net RX	185401946 Bytes

III. Existing Monitoring Frameworks

There are a number of open source Cloud Computing Tools and Resources available, some of them with an inbuilt monitoring module. For example, OpenNebula has a monitoring subsystem which captures CPU usage of the created VM, available memory and the Net data transmitted/received with the help of configured hypervisor drivers, in this case, Xen. Table II shows the sample output log from OpenNebula for a particular virtual machine. Net TX and RX shows total number of bytes transferred and received respectively.

Ganglia Monitoring System [10], initially designed for high performance computing systems such as clusters and Grids, is now being extended to Clouds, by the means of sFlow agents present in the Virtual Machines. Currently, sFlow agents [11] are available for XCP (Xen Cloud Platform) [12], Citrix XenServer [13], and KVM/libvirt [14] virtualization platforms. Nagios [15] is also one of the widely used network and infrastructure monitoring software application, for which some of the Cloud computing tools have provided hooks to integrate with.

Eucalyptus [16] is another open source cloud computing tool that implements IaaS private cloud, that is accessible via an API compatible with Amazon EC2 and Amazon S3 [17]. The monitoring service provided by Eucalyptus makes it possible for the guest virtual machines to be integrated with Nagios and Ganglia.

A. *Limitations in existing frameworks*

The monitoring metrics that OpenNebula collects are coarse grained and one may need to have fine grained process level data (e.g. CPU usage by netback driver process segregated into the ones used for different VMs [18], network bandwidth for a particular process, etc.) to incorporate appropriate QoS controls for some applications. In the future, OpenNebula developers have plans to use monitoring data as a feedback to scheduler (like Haizea [19]) to enforce placement policies [20].

Other Cloud computing tools like Eucalyptus, which integrate well with Ganglia and Nagios, also provide system level information, but at a more fine grained level. The gap here also remains the same when it comes to particular application level monitors.

Hence, we conclude that there is a need for unification of various software-hardware tools to be able to build an end-to-end framework which can bridge the gap between the existing frameworks and the required one. One such attempt is presented in Section IV.

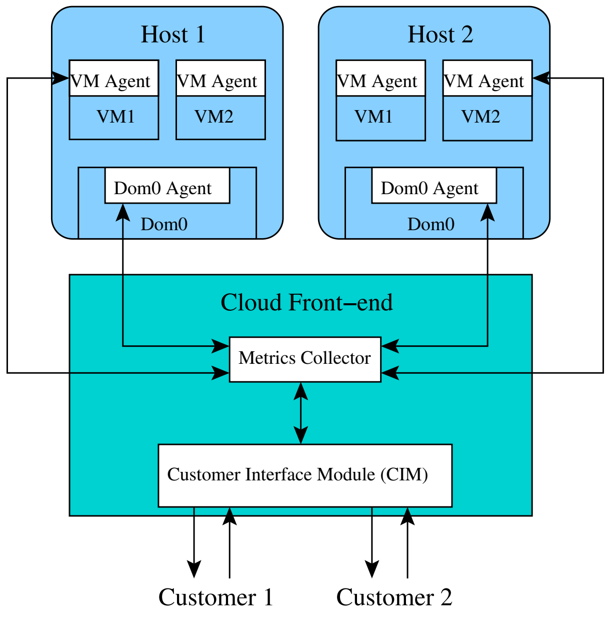


Fig. 3. Proposed Monitoring Framework Architecture

IV. Proposal : A Distributed Monitoring Framework

In this section, we propose a monitoring framework with monitoring agents distributed over various components in the Cloud. Next, we show monitoring results of the sample applications with our implemented monitoring framework.

A. Architecture

Figure 3 shows the basic architecture of a Distributed Monitoring framework. In a typical cloud setup, there could be a number of physical hosts (all of them running an independent hypervisor), and a front-end Cloud entity (like OpenNebula) to talk to external world. In our proposed architecture, each host carries a Dom0 agent and a number of VM agents (one for each VM). All of them communicate with the Metrics Collector (MC) placed inside the cloud front-end entity, which in turn, communicates with the Customer Interface Module (CIM).

Customers initiate the monitoring request by an interface provided by CIM. CIM instantiates the MC module. MC on-demand instantiates only those VM Agents and Dom0 Agent which need to gather monitoring information as requested by customers. The roles of each of these components is described below in detail:

1) VM Agent: It resides in VM, collects all VM specific metrics and passes it on to the Metrics Collector. VM specific metrics could be CPU, Memory and I/O Bandwidth utilization, either at the system level or fine-grained process level. Metrics Collector configures VM Agent, such that, it collates the required metrics. Most of the system level metrics could also be obtained by the Dom0 agent directly, except that process level metrics need a VM resident agent.

TABLE III
Metrics Specifications

Metric to monitor	Monitoring Interval(in ms)
CPU Usage in VM	500
CPU Usage (Dom0 contribution)	500
Incoming Network Bandwidth	1000
Outgoing Network Bandwidth	1000

2) Dom0 Agent: Dom0 Agent may also be called as Hyper Agent, since Dom0 is specific to Xen hypervisor. It resides in Dom0 in case of Xen, collects the per-VM effort that Dom0 incurs and forwards it to the Metrics Collector. As discussed earlier, Dom0 does a lot of processing on behalf of the guest VMs, which needs to be accounted to the corresponding VM. Hence, Dom0 agent complements the VM agent metrics in order to obtain the complete information. As an example, this could be the distribution of CPU usage contribution in the device driver process, virtual switch, or the netback driver, for each virtual machine.

3) Metrics Collector (MC): It collects the set of metrics, that are required by the customer, from the CIM; segregates the metrics required from each of the agents and configures the agents to obtain the same. Typical configuration could be the required monitoring metrics and the time interval after which it needs the monitoring data repeatedly.

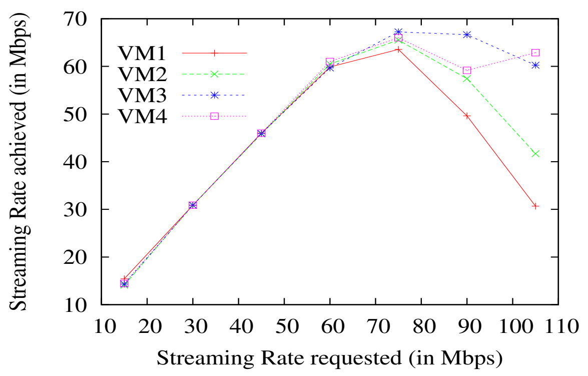
4) Customer Interface Module (CIM): Monitoring requirements for each customer could vary significantly. One may require very fine-grained details for debugging purposes or to take corrective actions at their end, others may leave it upto the cloud provider. CIM provides a great deal of flexibility for customers to customize the monitoring metrics based on their requirements.

B. Applications and Monitoring Results

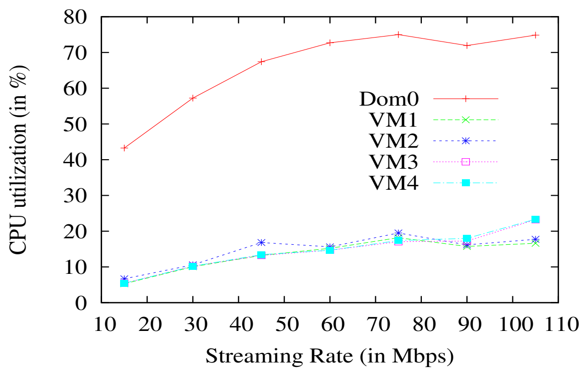
We choose three applications to demonstrate our monitoring framework capabilities: Video streaming, Encrypted video streaming, and httperf. All of the applications chosen gives a new dimension to our monitoring framework capability.

1) Video Streaming: We monitor a video streaming server hosted on VMs on the cloud. For this application, four VMs are deployed on OpenNebula cloud. VLC media player is used as streaming media server in all of the VMs to stream video to different clients based on the requests. Real time protocol (RTP) is used for streaming video over the network, since it is a standard for delivering audio and video over IP networks. In order to understand the dynamics of the streaming server resource usage, we use constant bit rate (CBR) streams in one instance of an experiment and vary the bit rate in next instance. CBR stream is generated by transcoding the variable bit rate (VBR) stream, by padding of artificial bits in between.

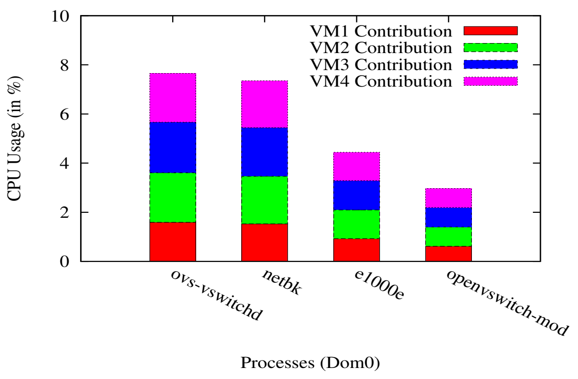
An example of a set of metrics a customer may want to monitor is shown in Table III. For realizing this requirement, VM agent and Dom0 agent use different tools and provide the relevant set of data to the MC, which in turn, forwards it to the CIM.



(a) Streaming rate throughput with varying requested rate



(b) CPU usage of Dom0 and VMs with varying requested rate



(c) CPU Usage for Dom0 processes per-VM as measured by Dom0 Agent at the requested streaming rate of 60 Mbps per VM

Fig. 4. Monitoring Results for Streaming Application

a) Bandwidth Monitoring: In our implementation, the VM agent uses the bwm-ng [21] tool for measuring input and output bandwidth utilization. Figure 4(a) shows the variation of achieved streaming rate with requested streaming rate. Requested streaming rate is referred to the streaming rate which client requests, or in other words, it the total bit rate of the video file(s) streamed. Achieved streaming rate is the actual streaming rate achieved measured by our VM agent.

b) CPU Usage Monitoring: Dom0 agent gathers Dom0 and VM CPU usage using XenMon tool. Figure 4(b) shows the CPU usage of Dom0 and four VMs while performing the test with different CBR streams. In contrast with previous httperf test, system saturates at a very high aggregate network bandwidth[1] of 240-300 Mbps. An explanation to this could be that, RTP applications typically use User Datagram Protocol (UDP) as the underlying protocol, which has comparatively less CPU overhead than TCP used in the httperf test.

Dom0 agent also calculates the CPU Usage distribution on a per-VM basis, as configured by MC. Dom0 agent calculates total number of pages mapped and unmapped by Dom0 on behalf of other VMs by capturing page_grant_map and page_grant_unmap events for all VMs during the httperf test. Since a guest VM always needs to keep buffers ready for the incoming packet, it offers pages to Dom0 to map onto its own address space. page_grant_map captures these map events. After the reception of the incoming packet by VM, Dom0 unmaps the page. The number of pages actually copied by Dom0 is approximately the same as number of map events as well as the number of unmap events, excluding the boundary conditions (for example, the number of pages that were already mapped at the time of start of the profiler and at the end of the profiler are assumed to be equal, unmap events that were pending at the start of the profiler and at the end of the profiler are also assumed to be equal, and so on). Hence, average of these two events gives us a rough approximation of the number of pages copied by Dom0 to the VM, as denoted by pages_copied[i] for ith VM in 1.

$$pages\_copied[i] \approx (map[i] + unmap[i])/2 \qquad (1)$$

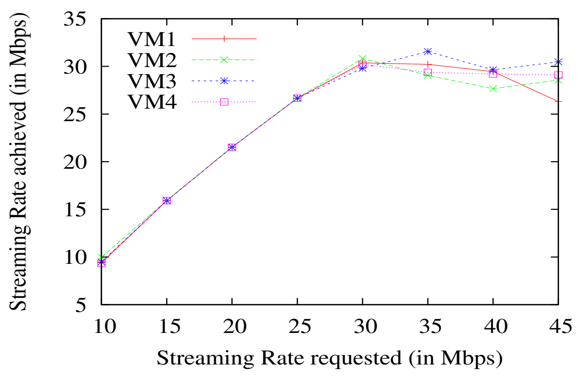
where, map[i] is no. of page_grant_map events for ith VM and unmap[i] is no. of page_grant_unmap events for ith VM.

$$cpu\_contribution\_ratio[j] = \frac{pages\_copied[j]}{\sum_i pages\_copied[i]} \qquad (2)$$

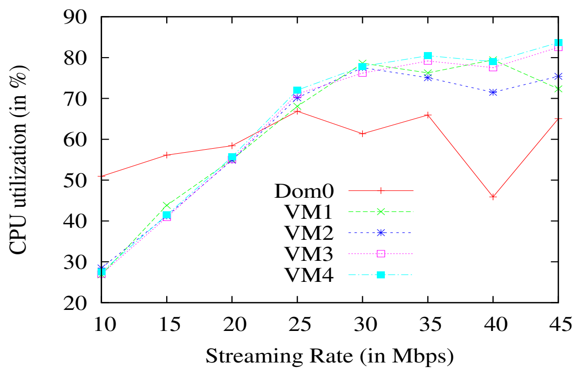
Using oprofile profiler [22], Dom0 agent calculates the CPU percentage used for a Dom0 process which does processing for other VMs and divide that in the ratio as calculated by cpu_contribution_ratio[j] for the jth VM in 2. Figure 4(c) shows the distribution of the CPU usage per-VM process level for streaming . It shows four processes running in Dom0 and their contribution towards each VM as calculated by above equations.

2) Encrypted Video Streaming: Next, we monitor the same video streaming application, but with on-the-fly encryption of the video and audio streams. We use Common Scrambling Algorithm (CSA) for encryption of the streams, as it is the most common algorithm used in Digital Video Broadcasting (DVB), popularly known as DVB-CSA. In our experiment, encryption is done purely in software by VLC media player. Since encryption is CPU intensive task, we expect to see high CPU usage of VM CPU. Figure 5(a) shows the variation of achieved streaming rate with requested streaming rate, and

[1]Aggregate network bandwidth refers to the summation of the saturation bandwidth of all VMs, namely VM1, VM2, VM3 and VM4.

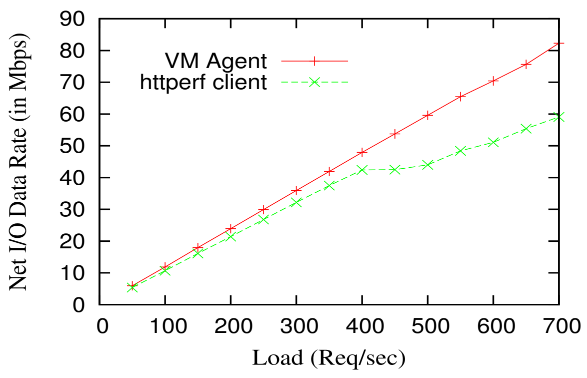


(a) Streaming rate throughput with varying requested rate

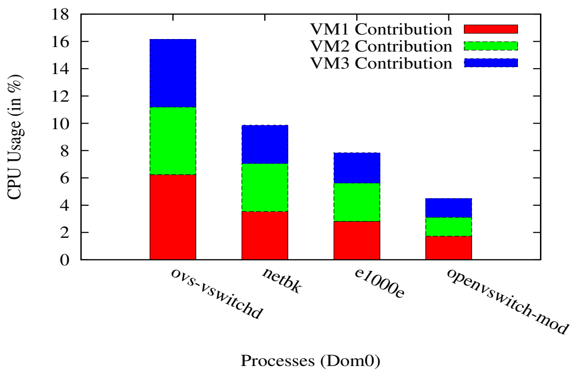


(b) CPU usage of Dom0 and VMs with varying requested rate

Fig. 5. Monitoring Results for Encrypted Video Streaming



(a) Bandwidth Monitoring by VM Agent and its comparison with Net I/O given by httperf for VM1



(b) CPU Usage for Dom0 processes per-VM as measured by Dom0 Agent at the load of 400 requests/sec

Metric to Monitor	*Total Allocation*		*Update Interval*	*Monitored Value*
	Speed	*No. of Cores*		
VM CPU usage	3.07 GHz	1	500 ms	**9.67 %**
Dom0 CPU usage contribution	3.07 GHz	1	500 ms	**14.31 %**
Incoming Network Bandwidth	100 Mbps		1000 ms	**3.18049 Mbps**
Outgoing Network Bandwidth	100 Mbps		1000 ms	**44.73642 Mbps**

(c) Metrics Monitored at the load of 400 requests/sec

Fig. 6. Monitoring Results for httperf Application

clearly indicates the saturation at just 30 Mbps streaming rate for one VM (or aggregate network bandwidth of 120 Mbps), reason for which is self-explanatory by the next figure. Figure 5(b) shows VM and Dom0 CPU usage variation with requested streaming rate. The key observation in this result is, VM CPUs becomes the performance bottleneck, leading to the system saturation. Dom0 processes' contribution towards each VM remains almost same in this case as of figure 4(c).

3) httperf: Let us now consider the httperf test application running at the customer's end. Along with the total VM and Dom0 CPU usage, we also monitor the CPU usage distribution of Dom0 processes on a per-VM basis. Figure 6(a) compares the bandwidth monitored by our VM agent with the Net I/O measured at the client end. As described earlier, Net I/O numbers provided by httperf at the client side correspond to the actual data transferred on TCP connections excluding headers and retransmissions, therefore, the actual output data rate by the Virtual Machines exceeds the Net I/O bandwidth measured by the client. Total VM and CPU usage graph is already depicted earlier in figure 2(c).

Figure 6(b) shows the distribution of the CPU usage per-VM process level for httperf, as calculated in 1 and 2. Figure 6(c) shows the monitored metrics as requested by the customer from the cloud provider at the load of 400 requests/sec. Monitored values are filled up dynamically by MC after gathering relevant information from different agents, at the time interval specified by the customer. In our example, Incoming Network Bandwidth, and Outgoing Network Bandwidth are collected by VM Agent and total VM CPU usage, and Dom0 CPU Usage contribution for each VM is collected by Dom0 Agent.

C. Discussion

There are a number of potential applications which could use monitoring data of infrastructural resources in clouds. One of them could be for scheduling decision of a new VM request by a client. Another application could dynamic reprovision resources based on monitoring feedback.

Let us consider the case when an existing VM finds its resources insufficient due to a new incoming requirement. We could reprovision the VM and Dom0 with more VCPUs, or place a new VM on a different host. In the streaming application, the system gets saturated because Dom0 CPU happens to be the bottleneck. Intuition suggests that allocating more VCPUs to Dom0 would prevent it from becoming a bottleneck. However this is not true because the ethernet driver used in our experiments executes in a serial fashion, hence it can't exploit parallelism provided by multiple cores. Since providing more VCPUs to Dom0 doesn't help, placing it on a new VM on a different host turns out to be a better decision in this case.

In the encrypted streaming application, the system gets saturated because VM CPU happens to be the bottleneck. Since the application is inherently multi-threaded, providing more number of VCPUs to the VM would prevent it from becoming a bottleneck. In contrast to the previous application, if an existing VM wants to scale its resources, reprovisioning the VM CPU is a better option here than to place a new VM on a different host.

In general, one can solve a system of equations to take the scheduling decisions numerically, based on the monitoring feedback. Further details are out of scope of this paper.

V. Conclusion

On a cloud infrastructure, having a monitoring framework for tracing the resource usage by a customer is useful in helping him to analyze and derive the resource requirements. Such frameworks also provide the transparency to the customer in knowing his actual usage. The proposed architecture provides a generic framework that can be customized as per the needs of the customers. It enables both provider and customer to monitor their application at a much finer granularity. Our future work would be to develop a closed loop framework, wherein the monitoring information would be used as feedback with proper controls in place, for meeting SLA requirements of customers.

References

[1] J. Lakshmi, "System Virtualization in the Multi-core Era - a QoS Perspective," Ph.D. dissertation, Supercomputer Education and Research Center, Indian Institute of Science, 2010.

[2] Vincent C. Emeakaroha and Marco A.S. Netto and Rodrigo N. Calheiros and Ivona Brandic and Rajkumar Buyya and Csar A.F. De Rose, "Towards autonomic detection of SLA violations in Cloud infrastructures," *Future Generation Computer Systems*, no. 0, pp. –, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11002184

[3] Jin Shao and Hao Wei and Qianxiang Wang and Hong Mei, "A Runtime Model Based Monitoring Approach for Cloud," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, july 2010, pp. 313 –320.

[4] Michael Armbrust, et al, "Above the Clouds: A Berkeley View of Cloud Computing," University of California, Berkeley, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, Tech. Rep., 2009.

[5] Mosberger, David and Jin, Tai, "httperf - a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, Dec. 1998. [Online]. Available: http://doi.acm.org/10.1145/306225.306235

[6] Alhamad, Mohammed and Dillon, Tharam and Wu, Chen and Chang, Elizabeth, "Response time for cloud computing providers," in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications &#38; Services*, ser. iiWAS '10. New York, NY, USA: ACM, 2010, pp. 603–606. [Online]. Available: http://doi.acm.org/10.1145/1967486.1967579

[7] D. Miloji andi and, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11 –14, march-april 2011.

[8] Chisnall, David, *The Definitive Guide to the Xen Hypervisor (Prentice Hall Open Source Software Development Series)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[9] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," HP Labs, http://www.hpl.hp.com/techreports/2005/HPL-2005-187.pdf, Tech. Rep., 2005.

[10] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817 – 840, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819104000535

[11] "Using Ganglia to monitor virtual machine pools, http://blog.sflow.com/2012/01/using-ganglia-to-monitor-virtual.html," 2012.

[12] "Xen Cloud Platform Project," 2012. [Online]. Available: http://xen.org/products/cloudxen.html

[13] "XenServer." [Online]. Available: http://www.xensource.com

[14] "Kernel Based Virtual Machine." [Online]. Available: http://www.linux-kvm.org/

[15] "Nagios." [Online]. Available: www.nagios.org/

[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.93

[17] "Amazon Elastic Compute Cloud," 2012. [Online]. Available: http://aws.amazon.com/ec2/

[18] L. Cherkasova and R. Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor." 2005 USENIX Annual Technical Conference, April, pp. 387–390.

[19] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[20] "Extending the Monitoring System," 2012. [Online]. Available: https://support.opennebula.pro/entries/352602-extending-the-monitoring-system

[21] "Bandwidth Monitor NG," 2012. [Online]. Available: http://sourceforge.net/projects/bwmng/

[22] J. Levon and P. Elie., "Oprofile: A system profiler for linux." [Online]. Available: http://oprofile.sourceforge.net