2014 IEEE International Conference on Services Computing

CLAMS: Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework

Khalid Alhamazani[1], Rajiv Ranjan[2], Karan Mitra[3], Prem Prakash Jayaraman[2], Zhiqiang (George) Huang [2], Lizhe Wang[4], Fethi Rabhi[1]
[1]School of Computer Science and Engineering, University of New South Wales
{ktal130, Fethir}@cse.unsw.edu.au
[2]CSIRO Computational Informatics
{rajiv.ranjan, prem.jayaraman, zhiqiang.huang}@csiro.au
[3]Luleå University of Technology, Skellefteå Campus, 93187 Skellefteå, Sweden
karan.mitra@ltu.se
[4]Chinese Academy of Sciences, Beijing, China
{lizhe.wang}@gmail.com

Abstract—**Cloud computing provides on-demand access to affordable hardware (e.g., multi-core CPUs, GPUs, disks, and networking equipment) and software (e.g., databases, application servers, data processing frameworks, etc.) platforms. Application services hosted on single/multiple cloud provider platforms have diverse characteristics that require extensive monitoring mechanisms to aid in controlling run-time quality of service (e.g., access latency and number of requests being served per second, etc.). To provide essential real-time information for effective and efficient cloud application quality of service (QoS) monitoring, in this paper we propose, develop and validate CLAMS—Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework. The proposed framework is capable of: (a) performing QoS monitoring of application components (e.g., database, web server, application server, etc.) that may be deployed across multiple cloud platforms (e.g., Amazon and Azure); and (b) giving visibility into the QoS of individual application component, which is something not supported by current monitoring services and techniques. We conduct experiments on real-world multi-cloud platforms such as Amazon and Azure to empirically evaluate our framework and the results validate that CLAMS efficiently monitors applications running across multiple clouds.**

Keywords- multi-clouds; cross-layer monitoring; QoS; cloud computing.

I. INTRODUCTION

Cloud computing is an emerging ICT service paradigm, that offers a flexible access to huge pool of resources such as processing, storage and network bandwidth with practically no capital investment and with modest operating costs proportional to the actual use (pay-as-you use model) [19]. A number of public cloud providers have emerged to be very successful in the recent past including Amazon Web Services (AWS), Microsoft Azure, Salesforce.com and Google App Engine. In cloud platforms, the infrastructure to deploy applications is enabled by virtualization technology also called Virtual Machine Monitor (VMM) [1]. Virtualization technologies are utilized in cloud platforms to run multiple instances of machines on the same physical machine and share its resources at the same time. Typically, these VMs are isolated and operate independently from each other. With virtualization, there is a host operating system and a guest operating system, the former runs on the physical machine whereas the latter runs on a virtual machine [2]. Virtualization isolates the VMs from each other, thereby making fault tolerant and isolated security context behavior possible. Most widely adopted virtualization technologies in the cloud computing paradigm include Xen, VMware, Hyper-V, KVM, and OpenVZ.

The advent of virtualization has led to the transformation of traditional data centers into flexible cloud infrastructure. With the benefit of virtualization, data centers progressively provide flexible online application service hosting [18] such as: web hosting, search, e-mails, and gaming. Largely, virtualization provides the opportunity to achieve high availability of applications in datacenters at reduced costs. The cloud platform is usually composed of several layers namely, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). For reliable and efficient management of application performance hosted on the *aaS layers, system administrators have to be fully aware of the compute, storage, networking resources, application performance and their respective quality of service (QoS) across the layers. QoS parameters (e.g, latency, renting cost, throughput, etc.) play an important role in maintaining the grade of services delivered to the application consumer and administrator as specified and agreed upon in the Service Level Agreement (SLA) document. The SLA guarantees scope and nature of an agreed QoS performance objective (also referred to as the QoS targets) that the cloud application consumer and administrators can expect from cloud service provider(s).

It is essential to note that the QoS parameters such as availability, application load, and throughput can vary in unpredictable ways depending on several factors (e.g., number of end-users connecting to application, physical resource or VM failure, VM overload etc.), making QoS monitoring an important task in ensuring the fulfillment of QoS targets (as specified in the SLA documents). Being aware of system's current software and hardware service status is vital to meeting QoS targets of cloud-hosted applications [9].

Based on the discussion above, it is obvious that cloud monitoring is an important task required for ensuring the QoS of applications hosted on the cloud platform [7] [8] [9] [10]. Primarily, monitoring is vital for: i) managing the QoS of software and hardware resources offered by the cloud; ii) providing continuous information on the status of resources to cloud providers and application administrators; and (iii) detecting and debugging software and hardware problems affecting applications' QoS. While there has been significant interest in the area of cloud monitoring by academia and industry, most of the existing approaches [7] [8] [9] [11] [12][13][14][15][16][17] suffers from several problems, for example, they are tightly coupled to a particular cloud platform and can only perform monitoring at a particular cloud layer (e.g., either IaaS or PaaS or SaaS). As with any distributed application hosting environments, there is a need for application deployment across multi-cloud environments to improve application resilience and benefit from economies of scale. Further, the QoS monitoring problem is cross-cutting as it extends across the multiple cloud service layers. For example, failure of VM (IaaS offering) affects the QoS of web application container (PaaS offering) or database application (PaaS offering) container hosted within that VM. This ultimately affects the QoS of end-user of that web application (SaaS offering). This exemplifies the need for cloud monitoring techniques and software frameworks that are capable of monitoring application components across layers and across multiple cloud provider environments. Hence, this renders the need for a uniform, extensive and effective multi-cloud, cross-layer monitoring framework that aid in controlling the application QoS based on real-time monitoring of the status of application components and underlying cloud platform (hardware and software).

We believe that a cloud provider can manage and administer the cloud-resources/applications more efficiently and effectively when he/she gains in-depth status information of the system and application components across layers individually as against a black box view. For example, an application such as WordPress[1] will typically have a MySQL database and an Apache Web Server as the underlying components. Current cloud-application monitoring frameworks like Amazon CloudWatch[2] typically monitor the entire virtual machine (hosting these components) as a black box. This means that the actual behavior of each application's component is not monitored separately. In this particular scenario, application monitoring is limited in scope where not all components that might be distributed across PaaS and IaaS layers are monitored. This limiting factor reduces the ability for fine grained application monitoring and QoS control across layers. Further, current cloud monitoring frameworks are mostly incompatible across multiple cloud providers. For example, Amazon CloudWatch does not allow monitoring application components hosted on non-AWS platforms. This defeats the distributed nature of cloud application hosting. These drawbacks trigger the significance of having interoperable and multi-layer enabled monitoring techniques and frameworks.

This paper addresses the key challenge of cross-layer multi-cloud monitoring. To this end, we propose Cross-Layer Multi-Cloud Application Monitoring as a Service (CLAMS) Framework. CLAMS framework enables monitoring applications across cloud layers as well as across heterogeneous cloud platforms. CLAMS allow efficient collection and sharing of QoS information across cloud layers. It employs a manger-agent approach that is cloud provider environments agnostic, i.e. making it compatible with any cloud provider. We present a proof-of-concept implementation of CLAMS monitoring application QoS across layers deployed and tested in multi-cloud environments such as Amazon AWS and Microsoft Azure. Using experimental analysis, we validate that CLAMS can efficiently monitor several cloud resources for multiple applications scenarios.

The rest of the paper is organized as follows. Section II presents the summary of current techniques and frameworks that support cloud monitoring. Section III presents a motivating scenario and overview of CLAMS framework. Section IV proposes CLAMS system framework for cloud applications monitoring. Section V presents implementation details. Section VI presents outcomes of experimental evaluation of CLAMS framework. Section VII concludes the paper.

II. RELATED WORK

In [20], Lattice monitoring framework is presented for monitoring virtual and physical resources. In this paper, a managed service is specified as a collection of Virtual Execution Environment (VEEs). Hence, Lattice is implemented to be able to collect information for CPU usage, memory usage, and network usage of each VEE and VEE host. Moreover, a dependable monitoring facility is presented in [21], called Quality of Service MONitoring as a Service (QoS-MONaaS). The focus of QoS-MONaaS approach is to: (i) continuously monitor the QoS statistics at the Business Process Level (SaaS); and (ii) enable trusted communication between monitoring entities (cloud provider, application administrator, etc.). Furthermore, a monitoring framework known as (PCMONS) is developed by incorporating previous frameworks and techniques [7]. PCMONS proves that cloud computing is viable way of optimizing existing computing resources in datacenters. Also, the paper notes that orchestrating monitoring solutions on installed infrastructures is viable. In contrast to above frameworks, CLAMS focuses on monitoring applications components across cloud layers as well as across heterogeneous cloud platforms.

In cloud platforms, recent efforts have been put into improving VMs monitoring and controlling. A number of frameworks have been proposed for VM management, which includes Simple Network Management Protocol (SNMP) for data retrieval. SBLOMARS [6] implements several sub-agents called *ResourceSubAgents* for remote monitoring. Each of SBLOMARS's sub-agents is responsible for monitoring a particular resource. Inside each of these sub-agents, SNMP is implemented for management data retrieval. In contrast to CLAMS which is focused on monitoring applications QoS in virtualized cloud computing environments, SBLOMAR focuses on enabling multi-constrain resource scheduling in grid computing environments.

In [3], CloudCop is a conceptual network monitoring framework implemented using SNMP. Basically, CloudCop adopts Service Oriented Enterprise (SOE) model. CloudCop

[1] https://wordpress.org/
[2] http://aws.amazon.com/cloudwatch/

framework consists of three components: Backend Network Monitoring Application, Agent with Web Service Clients, and Web Service Oriented Enterprise. While CloudCop focuses on network QoS monitoring, CLAMS is concerned with application QoS monitoring.

In [4], the authors propose a Management Information Base (MIB) called Virtual-Machines-MIB, to define a standard interface for controlling and managing VM lifecycle. It presents SNMP agents, which are developed based on NET-SNMP[3] public domain's agent. Besides read-only objects, Virtual-Machines-MIB provides read-write objects that enable controlling managed instances. To obtain the data of Virtual-Machines-MIB, mostly Libvirt[4] API and other resources such as VMM API are used [4]. While Virtual-Machines-MIB is concerned with monitoring IaaS-level (VM) QoS statistics, it does not cater for the QoS statistics of PaaS level application components.

In [5], the authors stress the importance to have a standardized interface for monitoring VMs on multiple virtualization platforms and this interface should be based on SNMP. The paper presents a framework for VMs monitoring which is fundamentally based on SNMP. The proposed work was built over three different VM hypervisors namely, VMware, Xen, and KVM. These three hypervisor were installed on two different OSs, which are MS Windows and Linux. Similarly to Virtual-Machines-MIB, this framework utilizes Libvirt API. Moreover, it implements an agent extension AgentX using Java. Primarily, this AgentX is to obtain VMs management data for the VMware, Xen, and KVM VMs and eventually the data is presented via web-based management. However, similar to [4], the approach given in [5] focuses on VM-level QoS monitoring, while completely ignoring application component level QoS management and monitoring.

In addition to the mentioned works above, libvirit-snmp is a subproject, which primarily provides SNMP functionality for libvirt. Libvirt-snmp allows monitoring virtual domains as well as it allows setting domain's attributes. Furthermore, Libvirt-snmp provides a simple table containing monitored data about domains' names, state, number of CPUs, RAM, RAM limit, CPU time.

III. MOTIVATION AND OVERVIEW

A. *Problem Discussion and Motivation*

A typical cloud application, for example, a multimedia content management system (CMS) as presented in Fig. 1 includes several components (that are deployed at PaaS level) such as media streaming server, web server, indexing server, database server, compute service, storage service and the underlying network. To achieve an end-to-end application monitoring for the CMS application, the monitoring technique needs to monitor QoS parameters e across cloud layers of CMS application stack including PaaS (e.g., web server, streaming server, indexing server, etc.) and IaaS (e.g., compute services, storage services, and network). Fig. 1 presents the QoS parameters that need to be monitored at each layer. The QoS parameters are presented and classified in Table 1 for the aforementioned CMS and other similar cloud applications (e.g., multi-tier web applications, content delivery networks, etc.) in general.

Furthermore, if monitoring needs to be carried across multiple cloud provider (typical of distributed mission critical applications) platforms such as Amazon AWS and Microsoft Azure, the cloud provider specific monitoring frameworks such as CloudWatch and Fabric Controller have limited functioning to work in multi-cloud environments e.g., CloudWatch on AWS cannot monitor application/resources hosted on Azure and vice-versa.

To achieve the above stated goal, there is a need for a cross-layer monitoring framework that has the capability to work across multiple cloud providers in a coordinated manner to delivery QoS requirements of distributed cloud applications.

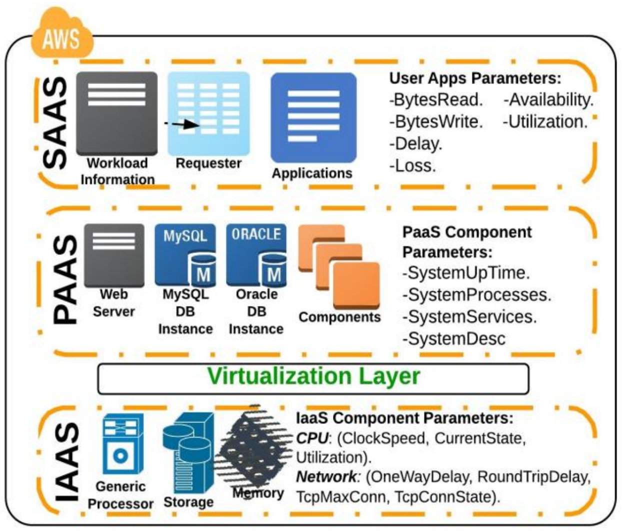


Figure 1: Multimedia Components and QoS Metrics Across the Cloud Layers

Table 1: QoS Targets for Relative Cloud Layers.

Cloud Layer	Layer Components	Targeted QoS Parameters
SaaS	User Applications (Servers App. Web App, Microsoft Word. etc)	BytesRead, BytesWrite, Delay, Loss, Availability, Utilization
PaaS	Web Server, Streaming Server, Indexing Server, Apps Server, etc	SystemUpTime, SysDesc, SystemProcesses, SystemServices
IaaS	Compute Services, Storage Services, Network, etc	CPU Parameters (Utilization, ClockSpeed, CurrentStte). Network Parameters (Capacity, Bandwidth, Throughput, ResponseTime, OneWayDelay, RoundTripDelay, TcpConnState, TcpMaxConn).

[3] http://www.net-snmp.org/
[4] http://libvirt.org/

B. *Overview*

Fig. 2 presents an overview of the philosophy driving the proposed CLAMS framework. As depicted in the figure, CLAMS employs an agent based approach for cross-layer, multi-cloud resource/application monitoring. In this multi-cloud approach, monitoring agents are deployed in various cloud provider environments based on application requirements. Each agent is responsible to monitor resource/application information at various layers including SaaS, PaaS and IaaS. A manager agent is responsible to collect QoS data from each monitoring agent.

IV. CLAMS: CROSS-LAYER MULTI-CLOUD APPLICATION MONITORING AS A SERVICE FRAMEWORK

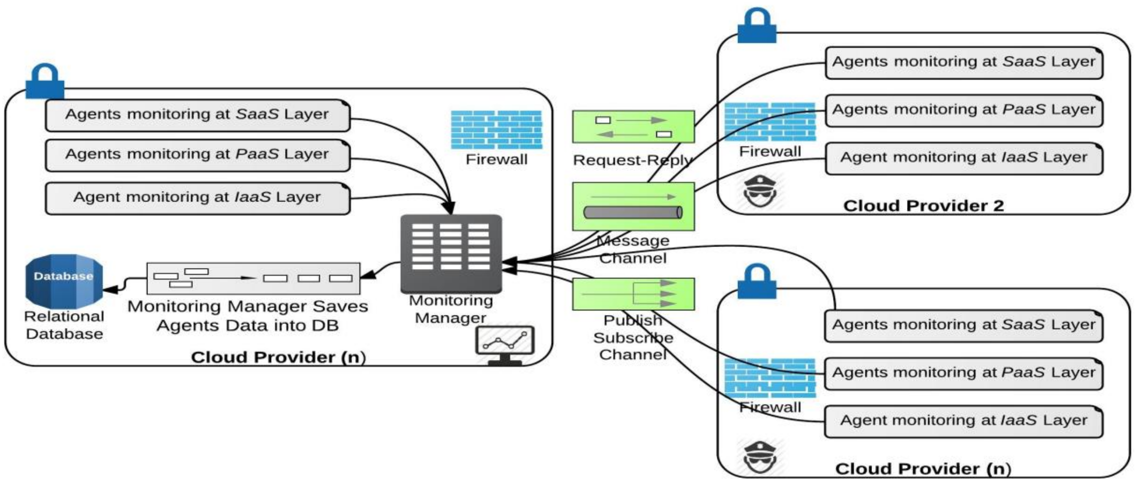
As mentioned in section I, in this paper we propose, develop and validate CLAMS, a novel approach for application(s) monitoring across layers on multi-cloud environments. CLAMS include mechanisms for efficient cloud monitoring at different *aaS layers. CLAMS provides standard interfaces and communication protocols that enable application/system administrator to gain awareness of the whole application stack across different cloud layers in heterogeneous, hybrid environments (monitor VMs hosted on different cloud platforms). In this way, CLAMS also satisfies the challenges related to interoperability between heterogeneous cloud resources. Fig. 3 presents a detailed architecture of the proposed CLAMS framework. The CLAMS framework comprises two main components namely, CLAMS Monitoring Manager and CLAMS Monitoring Agent.

A. *CLAMS Monitoring Manager*

The CLAMS Monitoring Manager is a software component that collects QoS information from CLAMS Monitoring Agents running on several virtual machines (VMs) across multi-cloud environments. In particular, the monitoring manager collects the QoS values from the agents running at the SaaS, PaaS and IaaS layers. The communication between the Monitoring Manager and the Agent can employ a push or pull technique. In case of pull technique, the manager polls the CLAMS agents at different frequencies, collects and stores the QoS statistics in a relational database (DB). When a push strategy is employed, the agents obtain the relevant QoS statistics and push the data to the Monitoring manager. As soon as the monitoring system is initialized on the cloud(s), the VMs running the CLAMS manager(s) and the agents boot up. Using discovery mechanisms like broadcasting, selective broadcasting or decentralized discovery mechanisms [24], the agents and manager discover each other. After discovering the address of each agent and manager, depending on the available strategy (push/pull) QoS statistics is collected by the manager from the agents. To illustrate further, consider a multimedia application running on the cloud where we have a media streaming server and an indexing server at the PaaS layer and storage server at IaaS layer.

Each component of the multimedia application is running and hosted on different VMs. Streaming server has an IP address say, 192.168.1.1, indexing server has an IP address 192.168.1.2, and the storage server has IP 192.168.1.3. Each VM also runs CLAMS monitoring agents that monitor applications and VM parameters. In this case, the manager can send first request to the agent on the streaming server VM specifying the IP address 192.168.1.1:8000 and stating the QoS target e.g., CPU utilization. Similarly, a second request is sent to the agent on the indexing server VM specifying the IP address (192.168.1.2:8000) and stating the QoS target e.g., Packets In. In the same way, a third request is sent to the agent on the storage server VM specifying the IP address (192.168.1.3:8000) and stating the QoS target e.g. actual used memory.

The CLAMS monitoring manager employs a QoS data collection schema to store QoS statistics into the local database and an agent schema to maintain the list of discovered agents. The CLAMS monitoring manager also



Agents monitoring at *SaaS* Layer
Agents monitoring at *PaaS* Layer
Agent monitoring at *IaaS* Layer
Firewall
Database
Relational Database
Monitoring Manager Saves Agents Data into DB
Monitoring Manager
Cloud Provider (n)
Request-Reply
Message Channel
Publish Subscribe Channel
Agents monitoring at *SaaS* Layer
Agents monitoring at *PaaS* Layer
Agent monitoring at *IaaS* Layer
Firewall
Cloud Provider 2
Agents monitoring at *SaaS* Layer
Agents monitoring at *PaaS* Layer
Agent monitoring at *IaaS* Layer
Firewall
Cloud Provider (n)

Figure 2: Overview of CLAMS Framework

incorporates an API that is used by other monitoring manager or external service to share the QoS statistics.

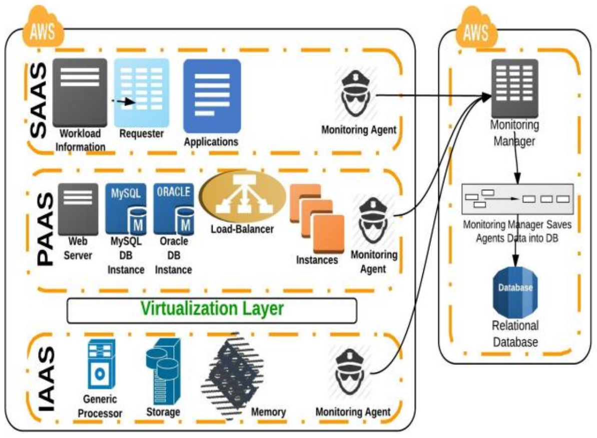


Figure 3: CLAMS Framework Architecture

B. CLAMS Monitoring Agent

Another major component of the CLAMS framework is the monitoring agent. The monitoring agent resides on VM running on the cloud and collects and sends QoS parameter values as requested by the manager. After the monitoring system initialization, the agent waits for the incoming requests from the manager or starts to push QoS data to the manager. Upon arrival of the request, the agent retrieves the stated QoS values belonging to a given process and/or a system resource and sends them back as a response to the Manager.

The monitoring agent has the capability to work in multi-cloud environments. Agent manager communication can be established using any approach that fits the application requirement e.g., publish- subscribe, client- server or Web services. It can also employ standardized protocols for communicating system management information like SNMP. The proposed blueprint does not restrict future developers from extending CLAMS to their purposes. In our proof-of-concept implementation explained in the next section, we have employed a combination of SNMP and RESTful Web services. The CLAMS monitoring agent also uses operating system dependent code to fetch corresponding application QoS statistics, for example, use of OS specific commands to get CPU usage in Linux and Windows systems.

C. CLAMS Hierarchical Support for Multi-Cloud Environments

As mentioned previously, the CLAMS monitoring framework is aimed to be agnostic of the underlying cloud platform i.e., the manager/agent may run on heterogeneous cloud platforms. In case the monitored framework is distributed across different cloud platforms e.g., between Amazon cloud platform and Windows Azure platform, then one manager and multiple agents will be residing on each of these cloud platforms. To achieve heterogeneity and multi-cloud functionality, a hierarchical approach can be applied using Super Managers as depicted in Fig. 4. The function of a Super Manager is marginally different from a monitoring manager. The Super Manager is responsible for coordinating between multiple monitoring managers using the monitoring manger's API. The monitoring managers (depicted as manager) as illustrated earlier will retrieve the monitored data from agents, and then they will re-send the data to the SuperManager. In a wider scope, a hierarchy of super manager can be formed where a SuperManager instance can collect data from multiple SuperManager instances, as shown in Fig. 4.

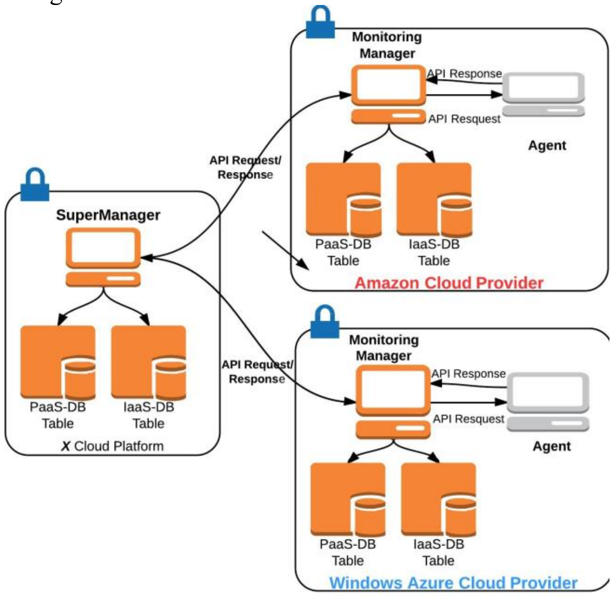


Figure 4: CLAMS Multi-Cloud Support

V. SYSTEM IMPLEMENTATION

The proof-of-concept implementation of the proposed CLAMS framework has been developed using Java and is completely cross-platform interoperable i.e., it works on both Windows and/or Linux operating systems. Fig. 5 presents proof-of-concept implementation screenshots.

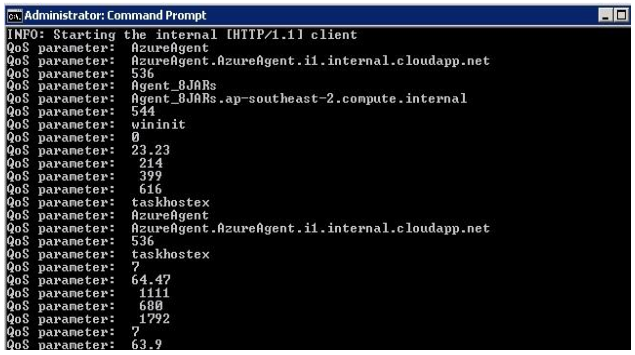
Monitoring Agent Implementation: The process of retrieving QoS targets is done by utilizing functionalities provided by SNMP, SIGAR[5] and other custom built APIs. For instance, SNMP is used to retrieve the QoS values related to networking, number of packets in and out, route information and, number of network interfaces. SIGAR is used to obtain access to low-level system information such as CPU usage, actual used memory, actual free memory, total memory and process specific information (e.g. CPU and memory consumed by a process). Moreover, network information such as routing tables can also be obtained using SIGAR. Both SIGAR and SNMP packages have their own operating system specific implementations to retrieve system information e.g. system resources, and user processes. To enable SNMP monitoring, we define new SNMP Objects

[5] http://www.hyperic.com/products/sigar

Identifiers (OIDs) in a sequence. For example function to get the CPU usage of a specific process (tomcat) is assigned an OID .1.3.6.1.9.1.1.0.0. Similarly, function to get process memory is assigned an OID .1.3.6.1.9.1.1.0.1.

Monitoring Manager Implementation: The monitoring manager uses a MySQL database to store the QoS statistics collected from the agents. For the proof-of-concept implementation, we used a pull approach where the Manager is responsible to poll for QoS data from agents distributed across multiple cloud provider VMs. The manager uses a simple broadcasting mechanism for agent discovery. On booting, a discovery message is broadcasted to known network. Agents that are available respond to the manager's request. The manager then records agent information to the agent database. The manager then starts off threads to query each agent in the agent database to obtain QoS parameters. The polling interval is a pre-defined constant and can be changed using the manager configuration files.

Agent Manager Communication: For the proof-of-concept implementation, the communication between the agent and the manager has been implemented using two techniques namely RESTful Web services and (SNMP). Having a RESTful approach enables easy lightweight communication between CLAMS agents and manager/super manager. Using a standardized SNMP interface makes CLAMS completely compatible with existing SNMP-based applications, tools and system and reduces the effort involved in collecting QoS statistics.



```
Administrator: Command Prompt
INFO: Starting the internal [HTTP/1.1] client
QoS parameter:  AzureAgent
QoS parameter:  AzureAgent.AzureAgent.i1.internal.cloudapp.net
QoS parameter:  536
QoS parameter:  Agent_8JARs
QoS parameter:  Agent_8JARs.ap-southeast-2.compute.internal
QoS parameter:  544
QoS parameter:  wininit
QoS parameter:  0
QoS parameter:  23.23
QoS parameter:   214
QoS parameter:   399
QoS parameter:   616
QoS parameter:  taskhostex
QoS parameter:  AzureAgent
QoS parameter:  AzureAgent.AzureAgent.i1.internal.cloudapp.net
QoS parameter:  536
QoS parameter:  taskhostex
QoS parameter:  7
QoS parameter:  64.47
QoS parameter:   1111
QoS parameter:   680
QoS parameter:   1792
QoS parameter:  7
QoS parameter:  63.9
```

Figure 5: CLAMS proof-of-concept Implementation

VI. EXPERIMENTS AND RESULTS

A. *Hardware and Software Configuration*

To evaluate the proposed CLAMS framework, experiments were conducted on Amazon AWS and Microsoft Azure platforms. We used standard small instances on each platform. The AWS instance has the following configurations: 619 MB main memory, 1 EC compute unit e.g. 1 virtual core with 1 EC2 compute unit, 160 GB of local instance storage, and a 64-bit platform. The Azure instance has the following configurations: 768 MB main memory, 1GHz CPU (Shared virtual core) and a 64 bit platform. During the execution of the experiment, we increased the number of AWS instances from 1 to 3 instances. Each virtual machine instance was running multiple CLAMS monitoring agents, each monitoring one or more processes at different *aaS layers. Further, VM's in the experiments were running Microsoft Windows Operating System. For persistent storage of monitoring agent and manager data, we use off storage volumes such as Elastic Block Store (EBS) in Amazon EC2 and XDrive in Windows Azure. Major advantages of architecting applications to adopt off instance storage are: i) each storage volume is automatically replicated, and this prevents data loss in case of failure of any single hardware component, ii) storage volumes offer the ability to create point in time snapshots, which could be persisted to the cloud specific data repositories.

B. *Experimental Setup*

The CLAMS system has two main components namely the Monitoring Manager and Monitoring Agent. Each agent comprises the corresponding SNMP and SIGAR package dependencies to accomplish the monitoring task. In the experiment, the monitoring manager triggers a request to monitoring agents, which in turn retrieve the requested QoS parameters from the hosted VM. Each agents running on the VM listens on a unique port e.g. VM1-IP:8000, VM1-IP:8001, enabling them to respond to queries from the monitoring manager independently. The agents send responses to the monitoring manager concurrently.

For experimental purposes and to demonstrate and validate CLAMS's cross-layer monitoring capability, each agent monitors several resources including system resources and user processes Table 2 presents the list of monitored processes/resources. On retrieving QoS data from the agents, the monitoring manager saves the data into the local database by classifying them as system performance or user applications QoS performance parameters.

Table 2: Monitoring across different layers

Process/Resource	**Description**	**Owner**
Tomcat7w.exe	Apache Tomcat 7	User
MySqld.exe	MySQL Workbench 6.0	User
Javaw.exe	Monitoring Manager	User
Lsass.exe	Local Security Authority Process	System
Winlogon.exe	Windows Logon App.	System
Services.exe	Services and Controller App.	System
VM CPU Usage	CPU usage of the entire VM	System
VM Memory Usage	Memory usage of the entire VM	System

Runtime Configuration: Monitoring agents as well as manager are packaged into jar files with corresponding dependencies and configured to run during VM boot process. The agents use a configuration file that specifies processes to monitor. Based on this information, at run-time, the agent determines the process id of the respective process. After finding the process id, the agent starts to retrieve specific QoS parameters for that process e.g. memory usage and CPU consumption.

Fig. 6 provides a detailed workflow of communication between the monitoring manager and agent. The monitoring manager instantiates parallel threads for each group of

Agents in one VM i.e., each thread is dedicated to only one VM to communicate with Agents running on that VM. Manager thread sends request to Agents addressed by IP address and port number. The request is for a list of QoS parameters monitored by the agent. After receiving the request, agents compute the QoS parameter values from the hosting VM. The agents then respond to the manager with corresponding QoS parameters.

To evaluate the proposed CLAMS framework, we deployed the agents and managers on four virtual machine instances (3 VM's on AWS and 1 on Microsoft Azure). On VM's that hosted the agent, depending on number of agents, the agents were bound to unique ports. E.g., if VM-3 hosted 30 Agents, it was bound to ports 8000-8030. Similarly if VM-4 hosted 10 agents, it was bound to ports 8000-8010.

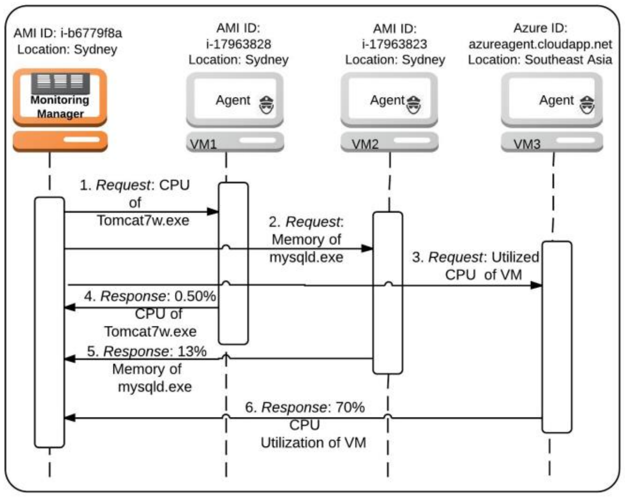

AMI ID: i-b6779f8a
Location: Sydney
AMI ID:
i-17963828
Location: Sydney
AMI ID:
i-17963823
Location: Sydney
Azure ID:
azureagent.cloudapp.net
Location: Southeast Asia
Monitoring Manager
Agent
Agent
Agent
VM1
VM2
VM3
1. Request: CPU of Tomcat7w.exe
2. Request: Memory of mysqld.exe
3. Request: Utilized CPU of VM
4. Response: 0.50% CPU of Tomcat7w.exe
5. Response: 13% Memory of mysqld.exe
6. Response: 70% CPU Utilization of VM

Figure 6: Manager/Agents run-time workflow

C. *Experimental Results and Discussion*

To validate CLAMS does not introduce significant overheads while monitoring QoS parameters across layers in multi-cloud environments, we ran experiments in 4 typical multi-cloud workload scenarios.

Scenario I: VM-1 hosts the Manager, VM-2 hosts 25 Agents, VM-3 hosts 30 Agents, and VM-4 hosts 30 Agents. In total, the manager communicates with 85 Agents deployed in multi-cloud environment (3 AWS instances and 1 Azure instance).

Scenario II: VM-1 hosts the manager, VM-2 hosts 10 agents, VM-3 hosts 20 agents, and VM-4 hosts 20 agents. In total, the manager communicates with 50 Agents.

Scenario III: VM-1 hosts the manager, VM-2 hosts 10 Agents, VM-3 hosts 10 Agents, and VM-4 hosts 10 Agents. In total the manager communicates with 30 Agents.

Scenario IV: VM-1 hosts the manager, VM-2 hosts 1 agent, VM-3 hosts 1 agent, and VM-4 hosts 3 agents. In total the manager communicates with 5 Agents.

For each scenario, we monitored the CPU and memory consumption of the monitoring manager. The result of the experiments is presented in Fig. 7 and 8. We computed the average CPU and memory utilization by the Manager for each scenario. Each evaluation scenario involving communication between agents and manager was run for duration of 30 minutes. The frequency of querying the agents for QoS parameters was set to 1 second.

The outcomes clearly indicate that the manager performance is stable with increase in the number of active agents. The CPU utilization grows up from 6.25% when manager is communicating with 5 Agents to 10.92% when the number of agents is 85. Likewise, memory consumed by the manager increased marginally from 177.5 MB with 5 agents to 177.85 MB with 85 agents. Moreover, we note, the manager or the agents during the experiment did not encounter any crash or malfunction. These outcomes clearly validate the resource efficient operation of the CLAMS framework and its ability and suitability to scale across multi-cloud environments.

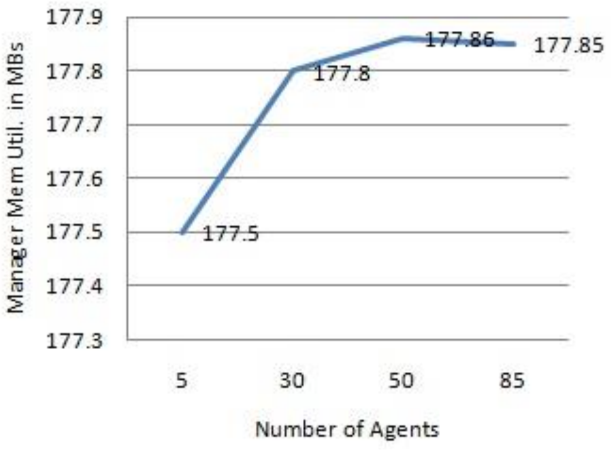


Figure 7: Manager Memory Utilization in MB

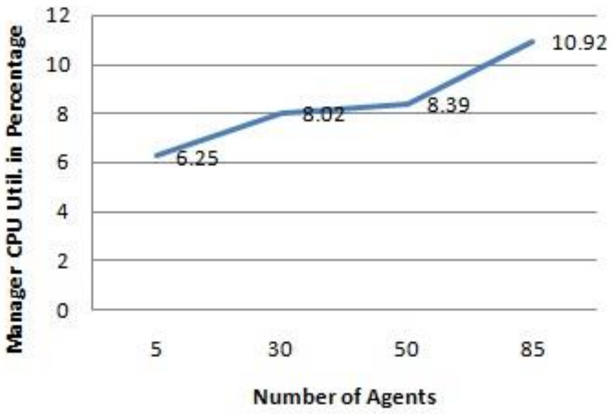


Figure 8: Manager CPU Utilization in Percentage.

In essence, we are motivated by the fact that there is a need for monitoring specific processes across cloud layers in multi-cloud environments. The proposed framework namely CLAMS demonstrates its capability to achieve this goal by enabling cross-layer monitoring in multi-cloud environments. Experimental evaluations of the CLAMS framework show a steady scalability of the monitoring manger while handling data from 5, 30, 50 and 85 agents simultaneously. Additionally, we note that the resource requirements of the CLAMS agent did not increase significantly when testing in environments with 5 and 85 agents. This further validates the CLAMS framework's ability to be a reliable, resource efficient cross-layer monitoring system that can scale across multiple cloud provider environments.

VII. Conclusion and future work

In this paper, we presented CLAMS—Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework. The novel features of CLAMS includes: (i) ability to monitor and profile QoS of applications, whose parts or components are distributed across multiple public or private clouds and (ii) ability to provide visibility into QoS of individual components of application stack (e.g., web server, database server). Our experimental evaluation study shows that proposed approach is feasible and does not have significant overheads.

Our future work will focus on extending the CLAMS in following aspects [22]: (i) we will develop Distributed Hash Tables (DHT) based decentralized messaging and indexing system for interconnecting Agents and Managers within CLAMS. For supporting scalable monitoring data query interface over DHT infrastructure, will implement additional data distribution and indexing techniques such as logical multi-dimensional or spatial indices. This decentralized approach will lead to better scalability and system performance as compared to the existing centralized architecture. The performance of the resulting decentralized CLAMS will be evaluated by measuring messaging latency, network traffic density, and additional message routing overheads and (ii) we will novel application workload behaviors and cloud resource QoS prediction models based on data collected from CLAMS. The prediction model will be based on the recent advances in computational statistical techniques [23] (e.g., time series clustering, decision tree learning, quadratic response surface models and Kernel Canonical Correlation Analysis.). The prediction models will capture that behavior of applications and its impact on overall QoS delivered by the underlying cloud services.

References

[1] S. N. T.-c. Chiueh and S. Brook, "A survey on virtualization technologies," *RPE Report*, pp. 1-42, 2005.

[2] M. Bolte, M. Sievers, G. Birkenheuer, O. Nieh√∂rster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt," *in Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 574-579.

[3] M. K. Nair and V. Gopalakrishna, ",CloudCop: Putting network-admin on cloud nine towards Cloud Computing for Network Monitoring," in *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*, 2009, pp. 1-6.

[4] R. Hillbrecht and L. C. E. d. Bona, "A SNMP-Based Virtual Machines Management Interface," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, 2012, pp. 279-286.

[5] Y.-S. Peng and Y.-C. Chen, "SNMP-based monitoring of heterogeneous virtual infrastructure in clouds," in *Network Operations and Management Symposium (APNOMS), 2011 13th Asia-Pacific*, 2011, pp. 1-6.

[6] E. Magana, A. Astorga, J. Serrat, and R. Valle, "Monitoring of a virtual infrastructure testbed," in *Communications, 2009. LATINCOM'09. IEEE Latin-American Conference on*, 2009, pp. 1-6.

[7] S. A. De Chaves, R. B. Uriarte, and C. B. Westphall, "Toward an architecture for monitoring private clouds," *Communications Magazine, IEEE,* vol. 49, pp. 130-137, 2011.

[8] B. Grobauer, T. Walloschek, and E. Stˆcker, "Understanding cloud-computing vulnerabilities," *IEEE Security and Privacy,* 2010.

[9] I. Brandic, D. Music, P. Leitner, and S. Dustdar, "Vieslaf framework: Enabling adaptive and versatile sla-management," *Grid Economics and Business Models,* pp. 60-73, 2009.

[10] J. Moses, R. Iyer, R. Illikkal, S. Srinivasan, and K. Aisopos, "Shared Resource Monitoring and Throughput Optimization in Cloud-Computing Datacenters," 2011, pp. 1024-1033.

[11] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: online monitoring and analytics for managing large scale data centers," in *Proceedings of the 7th international conference on Autonomic computing*, 2010, pp. 141-150.

[12] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST special publication,* vol. 800, p. 145, 2011.

[13] A. Letaifa, A. Haji, M. Jebalia, and S. Tabbane, "State of the Art and Research Challenges of new services architecture technologies: Virtualization, SOA and Cloud Computing," *International Journal of Grid and Distributed Computing,* vol. 3, 2010.

[14] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, "The characteristics of cloud computing," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, 2010, pp. 275-279.

[15] S. Zhang, S. Zhang, X. Chen, and X. Huo, "Cloud computing research and development trend," in *Future Networks, 2010. ICFN'10. Second International Conference on*, 2010, pp. 93-97.

[16] M. Ahmed, A. S. M. R. Chowdhury, M. Ahmed, and M. M. H. Rafee, "An Advanced Survey on Cloud Computing and State-of-the-art Research Issues," *International Journal of Computer Science Issues(IJCSI),* vol. 9, 2012.

[17] L. Atzori, F. Granelli, and A. Pescap√®, "A network-oriented survey and open issues in cloud computing," 2011.

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *ACM SIGCOMM Computer Communication Review*, 2008, pp. 75-86.

[19] R. Ranjan, K. Mitra, D. Georgakopoulos, "MediaWise Cloud Content Orchestrator", *Journal of Internet Services and Applications, Springer*, vol. 4, Jan 2013.

[20] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. M. Vaquero, K. Nagin, and B. Rochwerger, "Monitoring service clouds in the future internet," *Towards the Future Internet-Emerging Trends from European Research*, pp. 1-12, 2010.

[21] L. Romano, D. De Mari, Z. Jerzak, and C. Fetzer, "A Novel Approach to QoS Monitoring in the Cloud," 2011, pp. 45-51.

[22] K. Alhamazani, R. Ranjan, F. Rabhi, L. Wang and K. Mitra, "Cloud monitoring for optimizing the QoS of hosted applications," Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on , vol., no., pp.765,770, 3-6 Dec. 2012.

[23] A. Ganapathi et al., "Statistics-driven Workload Modeling for the Cloud," ICDE Workshops 2010, pp. 87-92, IEEE Computer Society.

[24] R. Ranjan; L. Chan; A. Harwood.; S. Karunasekera; R. Buyya., "Decentralised Resource Discovery Service for Large Scale Federated Grids," e-Science and Grid Computing, IEEE International Conference on , vol., no., pp.379,387, 10-13 Dec. 2007.