

Cross-layer multi-cloud real-time application QoS monitoring and benchmarking as-a-service framework

Author:

Alhamazani, Khalid

Publication Date:

2016

DOI:

<https://doi.org/10.26190/unsworks/18945>

License:

<https://creativecommons.org/licenses/by-nc-nd/3.0/au/>

Link to license to see what you are allowed to do with this resource.

Downloaded from <http://hdl.handle.net/1959.4/55964> in <https://unsworks.unsw.edu.au> on 2024-04-25

Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework

Khalid Twarish Alhamazani



A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computer Science and Engineering

The University of New South Wales

January 2016

The University of New South Wales
Thesis/Dissertation Sheet

Surname or Family name: Alhamazani

First name: Khalid

Other name/s: Twarish

Abbreviation for degree as given in the University calendar: PhD

School: School of Computer Science and Engineering

Faculty: Computer Science and Engineering

Title: Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework

Abstract 350 words maximum: (PLEASE TYPE)

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 and data processing frameworks) platforms with features such as elasticity, pay-per-use, low upfront investment and low time to market. This has led to the proliferation of business critical applications that leverage various cloud platforms. Such applications hosted on single/multiple cloud platforms have diverse characteristics requiring extensive monitoring and benchmarking mechanisms to ensure run-time Quality of Service (QoS) (e.g., latency and throughput). The process of monitoring and benchmarking cloud applications is as yet a critical issue to be further studied and addressed.

Current monitoring and benchmarking approaches do not provide a holistic view of performance QoS for distributed applications cross cloud layers on multi-cloud environments. Furthermore, current monitoring frameworks are limited to monitoring tasks and do not incorporate benchmarking abilities. In other words, there is no unified framework that combines monitoring and benchmarking functionalities. To gain the ability of both monitoring and benchmarking all under one framework will empower the cloud user to gain more in-depth control and awareness of cloud services.

The Thesis identifies and discusses the major research dimensions and design issues related to developing techniques that can monitor and benchmark an application's components cross-layers on multi-clouds. Furthermore, the thesis discusses to what extent such research dimensions and design issues are handled by current academic research papers as well as by the existing commercial monitoring tools.

Moreover, the Thesis addresses an important research challenge of how to undertake cross-layer cloud monitoring and benchmarking in multi-cloud environments to provide essential information for effective cloud applications QoS management. It proposes, develops, implements and validates CLAMBS: Cross-Layer Multi-Cloud Application Monitoring and Benchmarking, as-a-Service Framework. The core contributions made by this thesis are the development of the CLAMBS framework and underlying monitoring and benchmarking techniques which are capable of: i) 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 EC2, and Microsoft Azure); and ii) giving visibility into the QoS of individual application components, which is not supported by current monitoring and benchmarking frameworks. Experiments are conducted on real-world multi-cloud platforms to empirically evaluate the framework and the results validate that CLAMBS can effectively monitor and benchmark applications running cross-layers on multi-clouds.

The thesis presents implementation and evaluation details of the proposed CLAMBS framework. It demonstrates the feasibility and scalability of the proposed framework in real-world environments by implementing a proof-of-concept prototype on multi-cloud platforms. Finally, it presents a model for analysing the communication overheads introduced by various components (e.g. agents and manager) of CLAMBS in multi cloud environments.

Declaration relating to disposition of project thesis/dissertation

I hereby grant to the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or in part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all property rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstracts International (this is applicable to doctoral theses only).



Signature

Witness Signature

28/05/2016

Date

The University recognises that there may be exceptional circumstances requiring restrictions on copying or conditions on use. Requests for restriction for a period of up to 2 years must be made in writing. Requests for a longer period of restriction may be considered in exceptional circumstances and require the approval of the Dean of Graduate Research.

FOR OFFICE USE ONLY

Date of completion of requirements for Award:



In the name of Allah, the Most Merciful, the Most Compassionate. Recite and
your Lord is the most Generous - Who taught by the pen - Taught man that
which he knew not. **Surat Al-'Alaq (The Clot), the Holy Quran.**

Declaration

‘I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.’



Signature:

Date____28/05/2016_____

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 and data processing frameworks) platforms with features such as elasticity, pay-per-use, low upfront investment and low time to market. This has led to the proliferation of business critical applications that leverage various cloud platforms. Such applications hosted on single/multiple cloud platforms have diverse characteristics requiring extensive monitoring and benchmarking mechanisms to ensure run-time Quality of Service (QoS) (e.g., latency and throughput). The process of monitoring and benchmarking cloud applications is as yet a critical issue to be further studied and addressed.

Current monitoring and benchmarking approaches do not provide a holistic view of performance QoS for distributed applications cross cloud layers on multi-cloud environments. Furthermore, current monitoring frameworks are limited to monitoring tasks and do not incorporate benchmarking abilities. In other words, there is no unified framework that combines monitoring and benchmarking functionalities. To gain the ability of both monitoring and benchmarking all under one framework will empower the cloud user to gain more in-depth control and awareness of cloud services.

The Thesis identifies and discusses the major research dimensions and design issues related to developing techniques that can monitor and benchmark an application's components cross-layers on multi-clouds. Furthermore, the thesis discusses to what extent such research dimensions and design issues are handled by current academic research papers as well as by the existing commercial monitoring tools.

Moreover, the Thesis addresses an important research challenge of how to undertake cross-layer cloud monitoring and benchmarking in multi-cloud environments to provide essential information for effective cloud applications QoS management. It proposes, develops, implements and validates CLAMBS: Cross-Layer Multi-Cloud Application Monitoring and Benchmarking, as-a-Service Framework. The core contributions made by this thesis are the development of the CLAMBS framework and underlying monitoring and benchmarking techniques which are capable of: i) 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 EC2, and Microsoft Azure); and ii) giving visibility into the QoS of individual application components, which is not supported by current monitoring and benchmarking frameworks. Experiments are conducted on real-world multi-cloud platforms to empirically evaluate the framework and the results validate that CLAMBS can effectively monitor and benchmark applications running cross-layers on multi-clouds.

The thesis presents implementation and evaluation details of the proposed CLAMBS framework. It demonstrates the feasibility and scalability of the proposed framework in real-world environments by implementing a proof-of-concept prototype on multi-cloud platforms. Finally, it presents a model for analysing the communication overheads introduced by various components (e.g. agents and manager) of CLAMBS in multi cloud environments.

To My Parents

To My Family

To my loving memories of my grandfather and grandmother

Acknowledgement

Supervision Team: Rajiv Ranjan, Fethi Rabhi, Karan Mitra, and Prem Jayaraman

First and foremost, I would like to thank Allah for his blessings that allowed me to finish this thesis. I also would like to immensely thank my supervisors, Professor Fethi Rabhi and Associate Professor (Reader) Rajiv Ranjan, for their guidance, patience, financial support, and encouragement during the entire course of my PhD studies, especially during hardship times. Without them, this work would not have been completed. I also would like to acknowledge them for their valuable feedback, precious technical comments, and enduring encouragement to target top international conferences and journals. Their constant effort, motivation and advice have helped me face some of the tough challenges during the PhD years.

Special Thanks to my co-supervisors Prem Jayaraman and Karan Mitra who have been very supportive and helpful to me. I am thankful to them for their valuable comments and feedback during this work.

Finally, words would not be enough to express my sincere gratitude to my family. My parents, for their constant support, encouragement, and prayers to complete my studies.

List of Publications

- K. Alhamazani**, L. Wang, F. Rabhi, K. Mitra, and R. Ranjan, "Cloud monitoring for optimizing the QoS of hosted applications," in *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2012, pp. 765-770.
- K. Alhamazani**, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, *et al.*, "An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art," *Computing*, pp. 1-21, 2014.
- K. Alhamazani**, R. Ranjan, K. Mitra, P. P. Jayaraman, Z. Huang, L. Wang, *et al.*, "CLAMS: Cross-layer Multi-cloud Application Monitoring-as-a-Service Framework," in *Services Computing (SCC), 2014 IEEE International Conference on*, 2014, pp. 283-290.
- K. Alhamazani**, R. Ranjan, P. P. Jayaraman, K. Mitra, M. Wang, Z. G. Huang, *et al.*, "Real-time qos monitoring for cloud-based big data analytics applications in mobile environments," in *Mobile Data Management (MDM), 2014 IEEE 15th International Conference on*, 2014, pp. 337-340.
- K. Alhamazani**, R. Ranjan, P. P. Jayaraman, K. Mitra, C. Liu, F. Rabhi, *et al.*, "Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework," *IEEE Transactions on Cloud Computing*, 2015, no. 1, pp. 1, PrePrints PrePrints, doi:10.1109/TCC.2015.2441715

Tables Contents

1. INTRODUCTION	1
1.1. PREAMBLE	1
1.2. RESEARCH MOTIVATION	4
1.3. RESEARCH OBJECTIVES AND CONTRIBUTIONS.....	13
1.4. THESIS OUTLINE.....	17
2. CLOUD APPLICATIONS MONITORING, RESEARCH DIMENSIONS, DESIGN ISSUES	20
2.1. INTRODUCTION	20
2.2. CLOUD COMPUTING BACKGROUND AND OVERVIEW	21
2.2.1. <i>Resources Sharing</i>	25
2.2.2. <i>Resource Isolation</i>	25
2.2.3. <i>Resources Aggregation</i>	25
2.2.4. <i>Dynamics of Resources</i>	26
2.2.5. <i>Ease of Resource management</i>	26
2.3. CLOUD APPLICATION DEPLOYMENT.....	26
2.3.1. <i>Resource Selection</i>	27
2.3.2. <i>Resource Deployment</i>	28
2.3.3. <i>Resource Management</i>	29
2.3.4. <i>Resource monitoring and benchmarking</i>	29
2.3.5. <i>Resource auto-scaling</i>	30
2.4. CLOUD RESOURCE PROVISIONING	30
2.4.1. <i>Resource Provisioning</i>	31
2.4.2. <i>Application Provisioning</i>	32
2.5. APPLICATIONS MANAGEMENT CHALLENGES IN CLOUD ENVIRONMENT.....	35
2.5.1. <i>Resource Scenarios in Cloud environment</i>	35
2.5.1.1 Task submission	35
2.5.1.2 Workload management.....	36
2.5.1.3 Advanced reservations.....	36
2.5.1.4 Co-scheduling	36
2.5.2. <i>Resource Monitoring</i>	37
2.5.3. <i>Resource Benchmarking</i>	37
2.6. CLOUD MONITORING	37
2.6.1. <i>Monitoring Process</i>	37
2.6.2. <i>Monitoring, QoS, and SLA</i>	38
2.6.3. <i>Monitoring across Different applications, Cross-Layers, and Multi-Clouds</i> ..	40
2.7. CLOUD BENCHMARKING	43
2.7.1. <i>Benchmarking Definition</i>	43
2.7.2. <i>Benchmarking, QoS, and SLA</i>	44
2.7.3. <i>Cross-Layers Benchmarking on Multi-Clouds</i>	45
2.8. EVALUATION FRAMEWORK	47

2.8.1.	<i>Monitoring and Benchmarking Framework Architecture</i>	47
2.8.1.1	Centralized Architecture.....	48
2.8.1.2	Decentralized Architecture.....	50
2.8.2.	<i>Interoperability</i>	53
2.8.2.1	Cloud Dependent.....	54
2.8.2.2	Cloud Agnostic.....	55
2.8.3.	<i>Quality of Service (QoS) Matrix</i>	56
2.8.3.1	Single Parameter.....	56
2.8.3.2	Composite Parameters.....	57
2.8.4.	<i>Cross-Layer Monitoring and Benchmarking</i>	57
2.8.4.1	Layer specific.....	58
2.8.4.2	Layer Agnostic.....	58
2.8.5.	<i>Programming Interfaces and Communication Protocols</i>	60
2.8.5.1	Application Programming Interface.....	60
2.8.5.2	Command-Line Interface (CLI).....	62
2.8.5.3	Widgets.....	63
2.8.6.	<i>Communication Protocols</i>	63
2.9.	COMMERCIAL AND OPEN-SOURCE MONITORING AND BENCHMARKING TOOLS.....	64
2.9.1.	<i>Monitis</i>	65
2.9.2.	<i>RevealCloud</i>	65
2.9.3.	<i>LogicMonitor</i>	66
2.9.4.	<i>Nimsoft</i>	66
2.9.5.	<i>Nagios</i>	66
2.9.6.	<i>SPAE by SHALB</i>	67
2.9.7.	<i>Amazon CloudWatch</i>	67
2.9.8.	<i>OpenNebula</i>	68
2.9.9.	<i>CloudHarmony</i>	68
2.9.10.	<i>Windows Azure FC</i>	69
2.9.11.	<i>Lattice Framework</i>	69
2.9.12.	<i>QoS-MONaaS</i>	69
2.9.13.	<i>PCMONS</i>	69
2.9.14.	<i>SBLMARS</i>	70
2.9.15.	<i>Apache CloudStack</i>	70
2.9.16.	<i>Compuware's Gomez</i>	70
2.9.17.	<i>Cloud Object Storage Benchmark (COSBench)</i>	71
2.9.18.	<i>C-MART</i>	71
2.9.19.	<i>CloudGauge</i>	71
2.9.20.	<i>CLIQr</i>	72
2.9.21.	<i>Hawk-I</i>	72
2.9.22.	<i>mOSAIC Benchmark</i>	72
2.10.	CLASSIFICATION AND ANALYSIS OF CLOUD MONITORING AND BENCHMARKING TOOLS - GAP ANALYSIS.....	72
2.11.	SUMMARY.....	78
3.	CROSS-LAYER MULTI-CLOUD APPLICATION MONITORING-AS-A-SERVICE FRAMEWORK.....	79



3.1.	INTRODUCTION	79
3.2.	CLAMS: CROSS-LAYER MULTI-CLOUD APPLICATION MONITORING-AS-A-SERVICE FRAMEWORK	82
3.2.1.	<i>General Overview</i>	82
3.2.2.	<i>CLAMS Data Collection Model</i>	85
3.3.	CLAMS ARCHITECTURE COMPONENTS	87
3.3.1.	<i>CLAMS Monitoring Manager</i>	87
3.3.2.	<i>CLAMS Monitoring Agent</i>	90
3.3.3.	<i>CLAMS Super-Manager</i>	92
3.4.	VISIBILITY AND INTEROPERABILITY	95
3.4.1.	<i>CLAMS: Cross-layers monitoring (Visibility vs. Black Box View)</i>	95
3.4.2.	<i>CLAMS: Hierarchical Support for Multi-Cloud Environments (Interoperability)</i> 97	
3.5.	CLAMS APPLICATIONS SCENARIO	100
3.5.1.	<i>Big Data Analytics Application Scenario</i>	100
3.5.2.	<i>How we detect failures using a Conventional Approach</i>	102
3.5.3.	<i>Using CLAMS to detect and identify at which cloud layer a failure occurs</i> <i>(Visibility)</i>	103
3.5.4.	<i>Using CLAMS to detect and identify at which cloud platform a failure occurs</i> <i>(Interoperability)</i>	104
3.5.5.	<i>CLAMS Data Collection Model Scenario</i>	106
3.6.	CLAMS VS. OTHER MONITORING FRAMEWORKS	107
3.7.	SUMMARY	108
4.	CROSS-LAYER MULTI-CLOUD REAL-TIME APPLICATION QOS MONITORING AND BENCHMARKING AS-A-SERVICE FRAMEWORK	111
4.1.	INTRODUCTION	111
4.2.	CLAMBS: CROSS-LAYER MULTI-CLOUD APPLICATION MONITORING AS A SERVICE 113	
4.3.	CLAMBS ARCHITECTURE COMPONENTS	116
4.3.1.	<i>CLAMBS Architecture: Benchmarking Manager</i>	117
4.3.2.	<i>CLAMBS Architecture: Benchmarking Agent</i>	120
4.3.2.1	Workload Generator	120
4.3.2.2	Capabilities	121
4.4.	CLAMBS AND THE CHALLENGES OF QoS AND SLAs	122
4.4.1.	<i>CLAMBS for Unpredictable QoS parameters</i>	122
4.4.2.	<i>CLAMBS for Addressing SLAs Challenges</i>	124
4.5.	CLAMBS VS. BENCHMARKING FRAMEWORKS	125
4.6.	SUMMARY	131
5.	MODELLING AND IMPLEMENTATION OF CLAMS AND CLAMBS FRAMEWORK	132
5.1.	INTRODUCTION	132
5.2.	PROOF-OF-CONCEPT IMPLEMENTATION	133
5.2.1.	<i>Development Tools and Techniques</i>	133
5.2.1.1	JAVA	134

5.2.1.2	Eclipse	134
5.2.1.3	Apache Tomcat.....	135
5.2.1.4	Simple Network Management Protocol (SNMP).....	136
5.2.1.5	SIGAR (System Information Gatherer and Reporter).....	138
5.2.1.6	Restlet	138
5.2.1.7	JMeter	139
5.2.2.	<i>Cloud Platforms Used</i>	139
5.2.2.1	Amazon Web Services (AWS) Elastic Computing (EC2)	140
5.2.2.2	Microsoft Windows Azure Platform	140
5.2.3.	<i>CLAMBS: A Practical System Prototype</i>	140
5.2.3.1	CLAMBS Monitoring Agent Implementation	143
5.2.3.2	CLAMBS Benchmarking Agent Implementation	147
5.2.3.3	CLAMBS Manager Implementation	148
5.2.3.4	CLAMBS Agent and Manager Communication	151
5.3.	MODELING AND ANALYZING CLAMBS OVERHEADS IN MULTI-CLOUD ENVIRONMENTS	153
5.3.1.	<i>Abstract Model for CLAMBS framework Deployment</i>	155
5.3.2.	<i>Communication Overhead</i>	158
5.3.3.	<i>CPU load, Response and Search Time</i>	163
5.4.	SUMMARY	169
6.	EXPERIMENTATION AND EVALUATION	170
6.1.	INTRODUCTION	170
6.2.	HARDWARE AND SOFTWARE CONFIGURATION	172
6.3.	EXPERIMENTAL SETUP	174
6.3.1.	<i>CLAMBS Monitoring Agent Setup</i>	175
6.3.2.	<i>CLAMBS Benchmarking Agent Setup</i>	176
6.3.3.	<i>Runtime Configuration Monitoring Agent</i>	178
6.3.4.	<i>Runtime Configuration Benchmarking Agent</i>	180
6.4.	EXPERIMENTAL RESULTS AND DISCUSSION.....	181
6.4.1.	<i>CLAMBS Monitoring Agent</i>	181
6.4.2.	<i>CLAMBS Benchmarking Agent</i>	184
6.4.2.1	Data Download Latency Benchmark.....	184
6.4.2.2	Data Upload Latency Benchmark.....	185
6.4.2.3	Download/Upload Bandwidth Benchmark.....	186
6.5.	EXPERIMENTS SCENARIOS ANALYSIS FOR CLAMBS VALIDATION AND FEASIBILITY	187
6.5.1.	<i>Development Environment Limitations</i>	188
6.5.2.	<i>CLAMBS Manager Scalability under Benchmarking</i>	188
6.6.	SUMMARY	190
7.	CONCLUSION AND FUTURE WORK.....	192
7.1.	CONTRIBUTIONS OF THE THESIS WORK	192
7.1.1.	<i>Research questions</i>	192
7.1.2.	<i>Addressing First Research Question</i>	193
7.1.3.	<i>Addressing Second Research Question</i>	194

7.1.4.	<i>Addressing Third Research Question</i>	<i>195</i>
7.2.	LIMITATIONS	197
7.3.	FUTURE WORK	198
7.3.1.	<i>CLAMBS: Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service Framework.....</i>	<i>199</i>
7.3.2.	<i>Monitoring big data security and privacy.....</i>	<i>200</i>
	REFERENCES.....	202
	APPENDIX A: CLAMBS PROTOTYPE IMPLEMENTATION FILES	214
	A.1: CLAMBS MONITORING	214
	A.1.1: CLAMBS MONITORING AGENT SIGAR FUNCTIONS.....	214
	APPENDIX B: SNMP MIBS TREE	223
	APPENDIX C: BRAINSTORMING MIND MAP	224
	GLOSSARY	225

List of Figures

Figure 1.1: Cloud computing services layers.	3
Figure 1.2: ESA scenario.....	12
Figure 1.3: Thesis outline.	18
Figure 2.1: Cloud architecture and virtualization.	24
Figure 2.2: Cloud application deployment in cloud environment.	27
Figure 2.3: Provisioning and deployment sequence diagram.....	33
Figure 2.4: Components across cloud platform layers.	42
Figure 2.5: Framework network architecture.	48
Figure 2.6: Centralized monitoring/benchmarking framework architecture.	49
Figure 2.7: Decentralized monitoring/benchmarking framework architecture.	51
Figure 2.8: Interoperability classification.....	54
Figure 2.9: QoS matrix classification.	56
Figure 2.10: Components across cloud platform layers and QoS propagation.	59
Figure 2.11: Visibility categorization.	60
Figure 2.12: Different types of programming interfaces.	61
Figure 2.13: Evaluation Dimensions tree diagram.	75
Figure 3.1: CLAMS Framework overview.	83
Figure 3.2: CLAMS distributed components.	84
Figure 3.3: ER for the cloud layer, applications' components, and QoS parameters.....	87
Figure 3.4: CLAMS Monitoring Manager component and CLAMS Monitoring Agent components.	88
Figure 3.5: Interaction of the CLAMS Monitoring Manager and distributed Monitoring Agents – Pseudo Code.	90
Figure 3.6: The CLAMS Monitoring Agent startup and monitoring process – Pseudo Code. .	92
Figure 3.7: CLAMS hierarchical approach.	93
Figure 3.8: CLAMS Super-Manager hierarchy approach – wider scope.	94
Figure 3.9: Interoperable CLAMS components communication.	95
Figure 3.10: Cross-layers monitoring (Visibility).	97
Figure 3.11: Cross multi-clouds monitoring (Interoperability).	99
Figure 3.12: Applications components and QoS metrics cross-layers.	104
Figure 3.13: Applications components and QoS metrics across multi-cloud platforms.	105
Figure 3.14: ER for the cloud layer, applications' components, and QoS parameters.....	106
Figure 3.15: CLAMS – Cloud Monitoring Framework for cross-Layers applications components on multi-cloud Environments.....	109
Figure 4.1: Overview of CLAMBS framework.	114
Figure 4.2: CLAMBS distributed architecture.	116
Figure 4.3: CLAMBS framework Benchmarking Manager and Benchmarking Agent components.....	117
Figure 4.4: Visibility and interoperability of CLAMBS distributed components.....	123
Figure 5.1: UML based description of the CLAMBS framework.....	141
Figure 5.2: CLAMBS proof-of-concept Implementation.....	142
Figure 5.3: SIGAR CLAMBSMonitoringAgent.java – Code Snippet.....	145

Figure 5.4: SNMP CLAMBSMonitoringAgent.java – Code Snippet.	146
Figure 5.5: Screenshots of CPU and MEM Usage.	150
Figure 5.6: Screenshots of the Curve of CPU Usage.	151
Figure 5.7: Assigning unique port number for each CLAMBS Monitoring Agent – Snippet Code.	152
Figure 5.8: CLAMBS components communication based on RESTful and SNMP.	153
Figure 5.9: CLAMBS framework deployment class diagram.	157
Figure 5.10: Communications: 3 data centers, Manager  located on V _{2,2}	161
Figure 5.11: Communications: 3 data centers, Managers  located on V _{1,2} , V _{2,2} , and V _{3,1}	161
Figure 5.12: Different management structures for 17 CLAMBS Agents – Model 1.	164
Figure 5.13: Different management structures for 17 CLAMBS Agents – Model 2.	164
Figure 5.14: Different management structures for 17 CLAMBS Agents – Model 3.	165
Figure 6.1: Distributed CLAMBS components across datacenters.	173
Figure 6.2: CLAMBS Manager/Agents run-time communication workflow.	179
Figure 6.3: Manager CPU consumption in percentage (Monitoring Scenario).	183
Figure 6.4: CLAMBS Manager Memory utilization in MB.	184
Figure 6.5: Data Download Network Latency (Time in Seconds).	185
Figure 6.6: Data Upload Network Latency (Time in Seconds).	186
Figure 6.7: Download/Upload Bandwidth (Kilobytes per Seconds).	187
Figure 6.8: CLAMBS Manager memory consumption (benchmarking scenario).	189

List of Tables

Table 2-1: QoS parameters at each cloud platform layer.	43
Table 2-2: Summary of studied monitoring and benchmarking frameworks.....	77
Table 3-1: QoS parameters for relative cloud platform layers.	85
Table 3-2: QoS parameters for specific resources across cloud platform layers.....	106
Table 5-1: Illustration of the CLAMBS monitoring console output.	143
Table 5-2: Model analysis notation.	156
Table 5-3: Two scenarios for CLAMBS deployment layout.	162
Table 5-4: Messages communications overheads.	163
Table 6-1: Experiments objectives and evaluation.	172
Table 6-2: Monitoring various resources across different layers.	176
Table 6-3: Benchmarking parameters measurements parameters.	178
Table 6-4: Experimental workload scenarios.	181
Table 6-5: The experimental outcomes summary.	191

1. Introduction

1. 1. Preamble

“Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [109].” _____NIST

The cloud computing paradigm is shifting computing from physical hardware and locally managed environments to virtualized services [4]. Hence, this paradigm shift has the capability to reshape the Information Technology (IT) industry and it has been coined as the next revolution[49]. In other words, cloud computing is a commonly agreed name for an IT phenomenon. This phenomenon represents a significant change in the way IT services can be invented, developed, deployed scaled, updated, maintained, and also paid for [107].

Characteristics that describe cloud computing as summarized by the National Institute of Standards and Technology (NIST) are: on-demand self-service, ubiquitous network access, resource pooling, rapid elasticity and pay-per-use [99]. As shown in figure 1.1, cloud service types can be abstracted into three layers: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [62], [50], [182], and [71].

- *SaaS* – refers to the model in which applications are provided as a hosted service to cloud customers who access these services via the Internet.
- *PaaS* – is a cloud computing model to provide applications' components over the Internet. PaaS delivers hardware and software tools; mostly these tools are required for applications development.
- *IaaS* – provides access to fundamental computing, storage, and network resources in a virtualized environment.

For illustration, hardware and software resources form the basis for delivering IaaS and PaaS. The top layer focuses on application services (SaaS) by making use of services provided by the lower layers. PaaS resource and SaaS applications are often developed and provided by third party providers who are different from the IaaS providers [73]. To illustrate further, WordPress application, which is a SaaS layer resource, has two components: MySQL's Database, and Apache Tomcat Server. In this scenario, PHP, MySQL's Database and Apache Tomcat server can be a PaaS offering and integrated over Amazon EC (IaaS offering) to create the web application WordPress which is the (SaaS offering).

The term "Cloud Computing" has become widely used following the announcement of collaboration between IBM and Google in this field [65]. Cloud computing is composed of many technologies such as speed networks, a fast microprocessor, huge

memory, a high speed network and a reliable system architecture [65]. Virtualization, which is the fundamental element of cloud computing, changes the way physical resources are originally consumed. Virtualization is a software layer that manipulates the hardware to enable the sharing concept. The resultant services out of this sharing constitutes the cloud computing services [12].

Conceptually, the principle of cloud computing is not absolutely new; it is commonly considered the evolution of computing as a utility [184]. Utility computing was a vision stated 40 years ago, which refers to the desire that computing resources and services be delivered, utilized, and paid for as utilities such as water or electricity [171]. Throughout this thesis, I adopt the cloud computing definition provided by the National Institute Standards and Technology (NIST) as presented earlier.

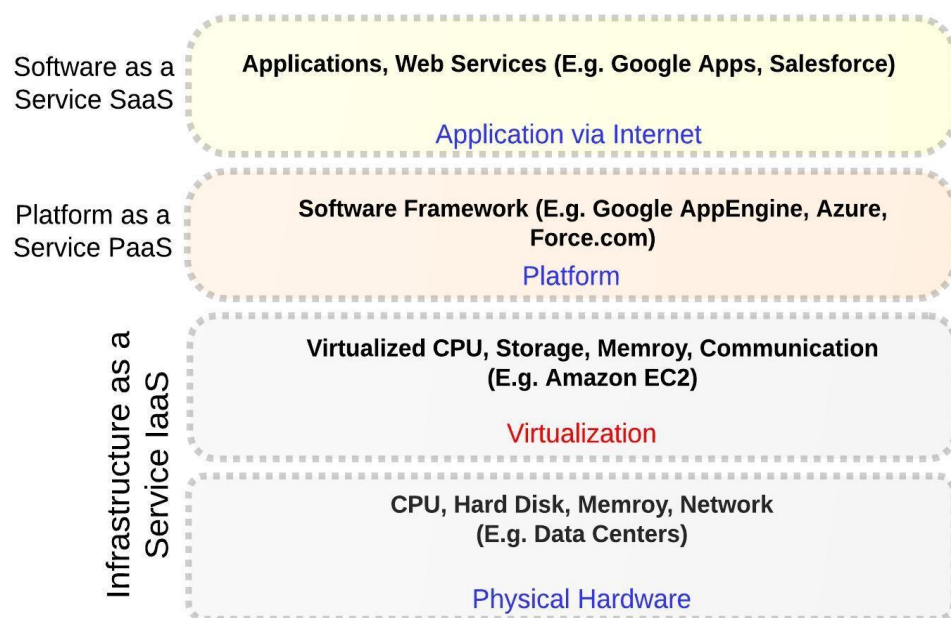


Figure 1.1: Cloud computing services layers.

1.2. Research Motivation

Cloud computing architecture consists of four deployment models that can be identified below [109] [189]:

- *Private Cloud* – The cloud platform is owned and managed by a private organization, which has an exclusive use of its infrastructure. This infrastructure can be installed by a third party such as VMWare and GoGrid.
- *Public Cloud* – The cloud platform is owned and managed by an organization, which sells cloud services to the general public or large industry groups (e.g. Amazon¹, Microsoft Windows Azure²).
- *Community Cloud* – The cloud platform provisions resources to a specific community/group that can be from different organizations for exclusive use. Such a community shares common concerns (e.g. Mission, security requirements, Policy). Government and healthcare are good examples of organizations that can leverage community cloud features.
- *Hybrid Cloud* – The cloud infrastructure is composed of two or more different types of clouds (e.g. Private, and Public) to perform distinct functions within the same organization. Furthermore, Cloudbursting is a practice of computing across multiple datacenters (internal and external). In other words, it is the ability of a cloud application to burst out of a private cloud into a public cloud as soon as the resources on the private cloud run out [93]. Also, Cloudbursting is growing in

¹ <http://aws.amazon.com/>

² <https://azure.microsoft.com/>

popularity because of the scalability and Pay-as-You-Go advantages offered by public clouds [81]. For example, a cloud user (e.g. a corporation) can run its services across private cloud infrastructure and Rackspace³ cloud.

The major concepts underpinning cloud architecture and cloud services are [180]:

- *Virtualization* – the software technology that hides the physical characteristics of cloud computing resources from the PaaS resources and SaaS-level application users by providing an abstract computing platform. It is a method whereby a single physical machine is distributed across autonomous and isolated software containers called virtual machines VMs [22]. A Virtual Machine Monitor (VMM) is placed instead of the operating system layer in the virtualization environment to manage the system resources across all available virtual machines.
- *Multi-tenancy* – an architecture in which a single instance of a software application serves multiple customers. In this scenario, each customer is called a tenant [111].
- *Web Service* –originally defined by the World Wide Web Consortium (W3C)⁴ as a software system designed to support interoperable machine-to-machine interaction over a network. Commonly, this refers to the ability of communication between clients and servers using the HTTP protocol over the Web and the Internet.

³ <http://www.rackspace.com/>

⁴ www.w3.org

In cloud platform, the web service interface enables the managing of virtual resources [127].

- *Service Level Agreement (SLA)* - SLA forms the services contract and defines particular aspects of the services (e.g. availability, performance, costs). In other words, SLA defines the minimal guarantees for cloud services to a customer [20]. For illustration, Amazon S3 (storage service) is offered under an SLA⁵ that states that Amazon will make commercially reasonable efforts to make S3 available with a monthly uptime percentage of at least 99.999% during any monthly billing cycle. Users are eligible to receive a service credit if Amazon fails to meet the SLA commitment.
- *Quality of Service (QoS)* – QoS provides an assurance level of supporting a resource's requirements for an application. A defined QoS is not limited to referring only to performance and availability, but also, to other aspects such as security and dependability [14].
- *Pay-as-You-Go (PAYG)* – Pay As You Go (PAYG) is a utility computing billing method which is applied in cloud computing [79]. PAYG enables cloud users to scale, customize and provision computing resources, including software, storage and development platforms. Cloud resources are charged based on used services.

As illustrated earlier, a cloud platform is usually composed of several layers namely, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-

⁵ <http://aws.amazon.com/s3-sla/>

Service (IaaS). Today, most of the heavily used applications are hosted on a cloud in one of such layers. For example, Google Mail service is a SaaS offering, Amazon SimpleDb (NoSQL Data Store) is a PaaS offering, Amazon Elastic Compute Cloud (EC2) is an IaaS offering [6], Google App Engine is a PaaS offering [66], and Salesforce's CRM an SaaS offering [150]. Cloud computing has transformed existing IT systems as services enabling the development of new and innovative applications. As more applications migrate into the cloud, reliable and efficient management of these application performance hosted on the *aaS layers is critical for the business. System administrators have to be fully aware of the computing, storage, networking resources, application performance and their respective 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 SLA document. The SLA guarantees the 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 application components availability, application load, and application throughput have a direct impact on application performance and can vary in unpredictable ways depending on several factors (e.g., number of application end-users connecting to application, physical resource or VM failure, VM overload etc.). Therefore, QoS monitoring is an important task in ensuring the fulfilment of SLA guarantees. QoS monitoring in this context refers to the continuous observing of the status of such parameters, which provides the required responsiveness of the whole monitored application. Being aware of the system's current software and hardware resource status is vital to meeting QoS targets of cloud-hosted applications [25]. Besides, such applications hosted on single/multiple cloud provider plat-

forms have diverse characteristics. For example, Multi-Media applications such as Content Delivery Network (CDN) will require high network throughput; whereas, E-Science applications such as ASKAP Radio Telescope [42] requires scalable storage to accommodate increasing amounts of data. Hence, monitoring the QoS characteristics is a challenging task.

Another obstacle facing the migration of applications to cloud is performance and predictability. Application migration can be rebuilding, redeploying, or re-hosting an application on a cloud platform. For illustration, an organization would re-deploy its web application on cloud platform for specific benefits such as scalability, automation, and improved development productivity. It is, therefore, challenging to inspect the status of such applications while re-hosting on a different platform. This may lead to the application's QoS parameters being in unwanted status without having the required awareness of their status. To address the performance and predictability of applications, benchmarking of cloud applications is being proposed. In a cloud environment, benchmarking refers to the process of defining the most suitable and testing a cloud resource (e.g. CPU, Memory utilization) [60]. Furthermore, benchmarking is applied to determine where improvements are required for specific QoS parameters.

However, performance unpredictability is the biggest obstacle facing the migration of applications (e.g., multi-layered business application, scientific data processing application, multi-media application, etc.) to clouds. While aforementioned applications are often held to strict QoS targets in terms of throughput, delay, or availability, little is known about the performance of applications in the cloud [152] [76], the response time variation induced by network latency, and cloud location. Since the QoS targets are encoded in legal SLA documents, determining these targets without understating the ap-

plication's performance is challenging. Hence, benchmarking applications can help in understanding the performance of applications when migrating to clouds.

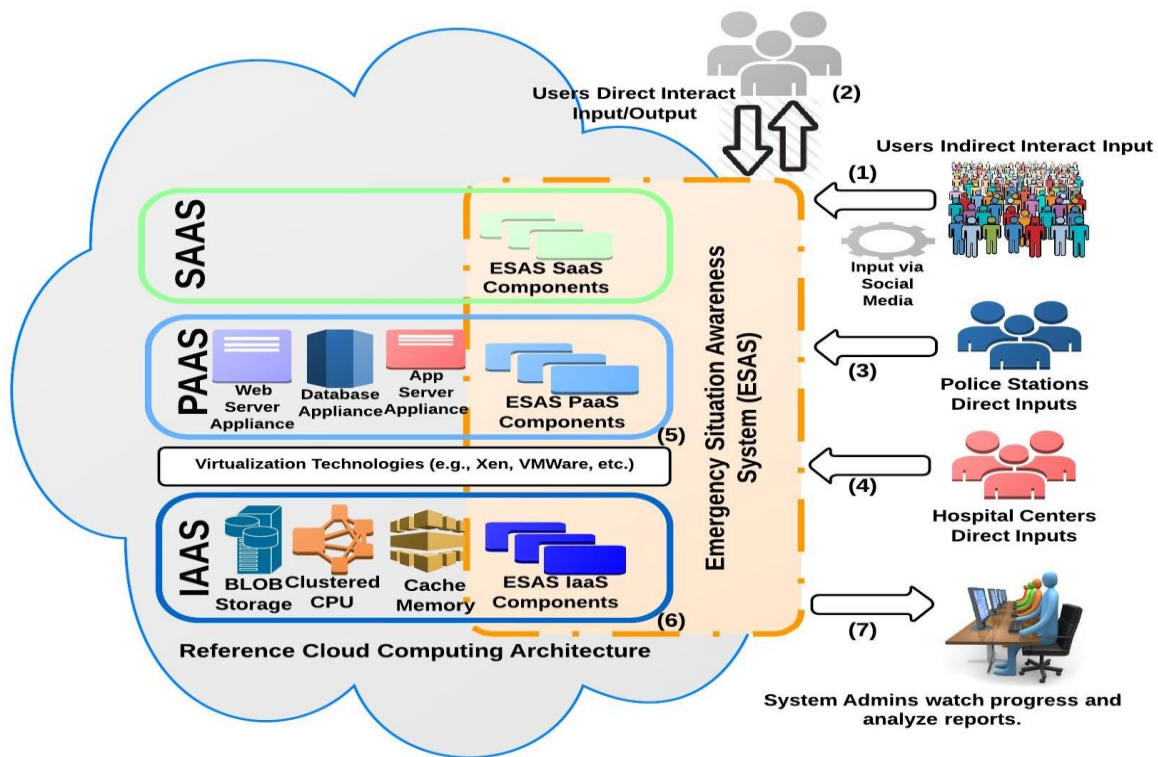
Moreover, it is not difficult to note that current SLA models supported by cloud providers are limited, since they do not cater for other complex QoS parameters which are generally associated with different applications types, including eResearch applications (e.g. data transfer latency, data transfer throughput, data security guarantee, data integrity guarantee etc.). For instance, the high-profile crash of Amazon EC2 cloud [9], which took down the applications of many SMEs, is a salient example of unpredictability in cloud environments. Some applications were down for hours, others for days. Moreover, in 2014, Windows Azure cloud virtual machines were down for a total of 42.94 hours globally [39]. In both cases, monitoring on the right time and event could save such resources from this failure. Also, benchmarking could provide an accurate inspection to the exact QoS parameter that caused that malfunctioning.

Identical resources provided by two different cloud providers may have various costs. This is one major factor where customer can opt resources from different cloud platforms. There are a number of available tools that can help cloud users to choose cloud resources. For instance, Clouddorado [183] calculates the price of IaaS-level CPU resources based on static features (e.g., processor type, processor speed, I/O capacity, etc.) [183]. Furthermore, different datacenter locations can be an important factor for consumers of resources. Along with the Pay-as-You-Go (PAYG) model, consumers of resources will have necessities and broad options to go for multi-cloud providers. Hence, based on aforementioned factors, a cloud user may have specific considerations and priorities preferences among such cloud resources' providers.

In the context of this thesis, I define multi-cloud hosting where applications components are distributed among multiple cloud providers' platforms. Therefore, enabling cloud users to monitor such distributed resources across different cloud platforms (referred as multi-clouds) is complex. Existing tools for monitoring and benchmarking cloud resources are mostly restricted to one provider and lack the ability to operate on multiple cloud platforms. For instance, CloudWatch which is provided by Amazon.com is limited to Amazon Elastic platform (EC2) [8]. Similarly, Microsoft Azure Fabric Controller (FC) is limited to work only on Azure platform [18]. In case of having distributed applications across multi-cloud platforms, the cloud user will need to have an independent monitoring and benchmarking tool for each platform. Because of using multiple tools, the users may face challenges in obtaining the holistic view of their applications' performance in a multi-cloud environment.

Another limitation of existing monitoring and benchmarking tools is that they only provide VM performance data. Moreover, they do not drill down into different layers of the application stack. In a cloud environment, the application stack is overlaid over multi-layers (e.g. IaaS, or PaaS). When hosting such distributed applications, users need to gain performance information for the whole application at different layers. Users will need to be able to monitor and benchmark the distributed application components. Current monitoring and benchmarking tools do not provide the ability to monitor or benchmark the distributed application components across different layers. For example, Cloudwatch will only provide data for the whole running VM instance but not for the applications' components within such VM [164]. Consider the illustration of an Emergency Situation Awareness (ESA) SaaS Application presented in figure 1.2. ESA is an

application that monitors mass gathering events in smart cities such as public demonstrations. Applications such as ESA are required to efficiently manage and respond to situations like public demonstrations, interior riot clashes, major festivals, and public/national events. Furthermore, such applications are comprised of many components deployed across the cloud layers. Thus, if users utilize existing limited tools, they will only gain data for the application as a whole or only performance for the hosting VM. As a result, the gained performance data is not adequate to ensure round-the-clock and robust operation of such ESA applications. Hence, cloud users require an in-depth



understanding of the application performance across the cloud layers.

Figure 1.2: ESA scenario.

Given the elasticity provided by cloud computing, a cloud platform can accommodate even unexpected changes in capacity by adding new instances of IaaS and PaaS resources (e.g., CPU, storage, network, database, etc.) and reducing them based on demand. The decisions to adjust capacity must be made frequently, automatically, and accurately to be cost effective. In addition to the elasticity complexity, failures or congestion of network links are sometimes inevitable, given the scale, dynamics, the crash or malfunction of a hardware resource, changes in workload patterns, or overloading of a hardware resource. Worse, hardware resource status can be changed intentionally through malicious external interference. Accordingly, to take advantage of cloud elasticity, there must be efficient mechanisms for monitoring and benchmarking cloud resources and applications' components. Also, knowing that cloud resources and application components can be distributed among multi-cloud platforms and across cloud layers triggers the need of having monitoring and benchmarking that are able to capture such distributed applications' components individually.

Being aware of application performance through continuous monitoring and benchmarking will enable the user to manage and support any unwanted changes. Consequently, cloud application provisioning and auto-scaling process could be made more effective to guarantee the applications' SLA. For illustration, application provisioning, auto-scaling, monitoring, SLA, and QoS performance are all interconnected and can cause an impact on each other. After application provisioning and deployment, continuous monitoring is the key process which can trigger any required auto-scaling process based on the application QoS performance. But, to detect the required application performance, both cloud resources' providers and customers need the SLA to define QoS parameters.

Determining how an application's performance will be impacted due to unpredictable conditions is a complex problem and is the foundational research motivation for this PhD.

1.3. Research Objectives and Contributions

Uncertainties in cloud environments can be tackled through the development of effective, scalable, inter-operable, easy-to-use monitoring and Benchmarking techniques.

Accordingly, this PhD thesis, focuses on developing techniques and frameworks for effective multi-layer monitoring and benchmarking of application performance deployed on multi-clouds. This monitoring and benchmarking framework enables capturing the QoS parameters that define the applications' performance characteristics. Furthermore, it also investigates technical challenges and dimensions pertaining to cloud application engineering that have direct impact on the application's performance and QoS.

In particular, it investigates a model that enables monitoring of applications' components and the associated QoS parameters to detect application performance variations under uncertainties, which refers to the non-estimated actions or events. Furthermore, the model is extended to enable real-time benchmarking of cloud-hosted applications.

In summary, the aim of this PhD research is to *investigate, propose, develop, implement and validate novel techniques and frameworks for real-time cross layer monitoring and benchmarking of QoS parameters of applications deployed on multi-cloud environment.*

To address the aim stated above, the following three research questions were formulated:

1. What is the current state-of-the-art architecture dimensions and issues of cloud applications' monitoring and benchmarking? In particular:
 - What is the body of the knowledge in current cloud monitoring and benchmarking tools and techniques?
 - What is the support for multi-cloud and cross layers monitoring and benchmarking?
2. How to design a monitoring tool which is scalable, dynamic, agnostic to cloud platform, agnostic to cloud layer, and agnostic to cloud application type? In particular:
 - How to determine layer specific application monitoring requirements; i.e., how cloud consumers can stipulate at which cloud layer (SaaS or PaaS or IaaS) his/her application should be monitored?
 - How cloud consumers can stipulate on which cloud provider platform or datacentre his/her application should be monitored?
 - How to model QoS and SLA information to monitor applications' performance?
3. How to design a benchmarking tool which closely integrates with a monitoring tool and is able to perform real-time benchmarking of applications' components at SaaS, PaaS, and IaaS layers?

The thesis makes the following contributions which addresses the above research questions:

In relation to research Question 1,

- Advancing the fundamental understanding of cloud resource and application monitoring and benchmarking concepts.
- Identification of the main research dimensions for developing monitoring and benchmarking techniques pertaining to multi-cloud hosted multilayers components applications.

By addressing research question 2 and 3, the thesis contribution is developing cross-layer cloud monitoring and benchmarking techniques for multi-cloud environments. In particular, it proposes the Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service (CLAMBS) Framework. CLAMBS has the following novel features:

- It provides the ability to monitor and profile the QoS of applications, whose components are distributed across multiple heterogeneous public and/or private clouds;
- It provides visibility into the QoS of individual components of application stack (e.g., CPU at IaaS layer, Database server at PaaS layer, and web application at SaaS layer). In particular, CLAMBS facilitates efficient collection and sharing of QoS information across SaaS, PaaS, and IaaS layers by deploying a cloud provider agnostic intelligent multi-agent technique;

- It provides benchmarking-as-a-service that enables the establishment of baseline performance of application deployed across multiple layers using a cloud-provider agnostic technique; and
- It is a comprehensive framework allowing continuous (real-time) benchmarking and monitoring of multi-cloud hosted multi-layered applications.

Furthermore, to verify, validate and evaluate the proposed CLAMBS framework, the thesis:

1. Implements the Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service (CLAMBS) Framework in Java, SNMP, RESTlet technology, and SIGAR. The total number of code lines was almost 4000 lines.
2. Demonstrates the scalability and efficiency of CLAMBS by conducting extensive real-world experimentations on cloud platforms such as Amazon AWS, and Microsoft Azure platforms.
3. Presents an empirical evaluation of CLAMBS framework.

The proposed techniques and frameworks can help system administrators and applications developers as follows:

- (i) keeping the cloud services and applications operating at peak efficiency;
- (ii) detecting variations in service and application performance;
- (iii) accounting the SLA violations of certain QoS parameters; and
- (iv) tracking the leave and join operations of services due to failures and other dynamic configuration changes.

It should however be noted that the developed framework is quite generic, and agnostic to cloud platforms, applications, service and cloud layers.

1.4. Thesis Outline

The thesis is organized into 7 chapters. Figure 1.3 presents an illustration of the thesis structure.

Chapter 2 presents a literature background to cloud applications life-cycle, resources provisioning, applications monitoring, and monitoring research and evaluation dimensions. Moreover, it discusses the QoS and SLA in cloud environments. Furthermore, this chapter provides a discussion about cloud resources benchmarking. In addition, a literature review of monitoring approaches identifying the need for monitoring across different cloud applications and layers as well as monitoring in multi-cloud environments is presented. Finally, it contains a classification and analysis for monitoring and benchmarking tools based on the novel taxonomy. This chapter is based on a published paper (K Alhamazani et al., 2014a).

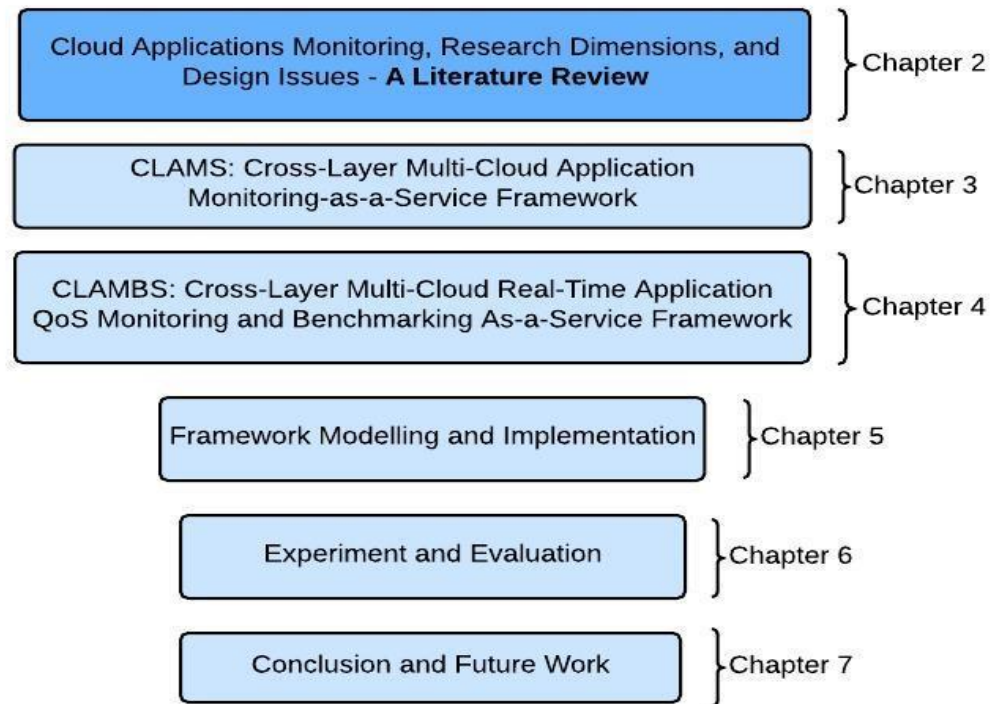


Figure 1.3: Thesis outline.

Chapter 3 presents in-depth discussion of the proposed applications monitoring across cloud application and layers. The proposed framework is Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework (CLAMS). This chapter is based on a published paper (K Alhamazani et al., 2014b, K Alhamazani et al., 2014c). The system framework primarily targets the applications' components distributed across cloud layers and platforms.

Chapter 4 introduces a new mechanism for application performance on multi-clouds environments. The proposed system CLAMBS extends the CLAMS framework proposed in chapter 3. The proposed framework is called Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework. The ex-

tension aims to apply benchmarking tasks across multi-cloud data-centers. This chapter is based on my published paper (K Alhamazani et al., 2015).

Chapter 5 presents a performance model and implementation of the proposed CLAMBS framework. This chapter is based on my published paper (K Alhamazani et al., 2015). This chapter mainly studies the scalability of CLAMBS in terms of communication messages complexity and overheads within CLAMBS framework. Moreover, this chapter presents an implementation of the proposed system frameworks CLAMS and CLAMBS. It provides detailed discussion of implementing the proposed frameworks in a real-world environment. Finally, in this chapter the details of JAVA implementation and the other harnessed technologies are presented.

Chapter 6 presents extensive evaluations and experimental results of the system presented in Chapter 3, and Chapter 4 respectively. Real-world experimentations to prove the feasibility of CLAMS are performed, supported by real-world outcomes to validate the efficiency and scalability of the proposed framework. Similarly, the chapter presents and discusses the results of extensive evaluation of CLAMBS in a real-world environment.

Finally, chapter 7 concludes the thesis with pointers to possible future works. This chapter is based on published papers (K Alhamazani et al., 2014b, K Alhamazani et al., 2015). In this thesis, the term CLMABS framework is used to refer to both CLAMS and CLAMBS.

2. Cloud Applications Monitoring, Research Dimensions, Design Issues

2.1. Introduction

A major focus of this thesis is to address the challenges of application monitoring and benchmarking in cross-layers and multi-cloud environments. Monitoring is managing the performing condition of software and hardware resources in a cloud environment. It provides data about the status/health of the monitored resource; for example, the CPU and memory utilization for the application deployed in cloud platform. Benchmarking is done to test and estimate the application's performance before the cloud application deployment. In a cloud environment, the process of application provisioning is to effectively manage the configuration and deployment of applications in cloud environments. Provisioning of applications in cloud computing environments is quite challenging considering the large number of heterogeneous cloud resources e.g. VM configurations and QoS parameters (e.g. CPU, memory, and network I/O). Traditionally, the term *resources* means denoting a physical entity, such as a computer, network, or storage. But, in this thesis the term *resources* is used in a generic sense, to indicate any capability that may be shared in a cloud environment (e.g. physical resources and virtualized resources).

This chapter presents the current state-of-the-art in cloud applications monitoring and benchmarking. An evaluation framework that identifies the research dimensions and design issues in the current state-of-the-art is developed. The key challenges in developing cross-layer multi-cloud application performance monitoring and benchmarking tools are highlighted. Finally, the chapter presents a comparative analysis and identifies the gaps in the current state-of-the-art.

This chapter is organized as follows. Section 2.2 presents background on cloud computing and its key technologies. Section 2.3 explores the aspects of cloud application deployment. Section 2.4 presents some background on cloud resource provisioning. Section 2.5 discusses and explores the challenges of application management in cloud environments. Section 2.6 provides a definition of cloud monitoring and identifies the respective challenges. Likewise, section 2.7 presents cloud benchmarking and the related challenges. In section 2.8, the evaluation framework for monitoring and benchmarking architecture in cloud environments is developed. In section 2.9, the current state-of-the-art in commercial and open-source monitoring and benchmarking tools and approaches is presented. Section 2.10 presents classification and analyses cloud monitoring and benchmarking tools against the evaluation framework presented in section 2.8. Finally, section 2.11 concludes the chapter.

2.2. Cloud Computing Background and Overview

The elasticity, pay-as-you-go model and low upfront investment offered by clouds, have led to a proliferation of application providers. For example, popular applications such as Netflix and Spotify use clouds such as Amazon EC2 to offer their services to millions of

customers worldwide. The success of cloud computing can be attributed to virtualization that enables multiple instances of virtual machines (VMs) to run on a single physical machine via resource (CPU, storage and network) sharing, thereby leading to flexibility and elasticity, as new instances can be launched and terminated as and when required.

Virtualization is the key technology for cloud computing success and popularity. As illustrated in chapter 1, cloud computing is divided into three layers, namely, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). The underlying physical machine is generally provided in the form of computing clusters, grids or individual servers. Figure 2.1, presents the cloud platform architecture and the virtualization concept. In this figure, the physical hardware resources are isolated by the virtualization layer. The hardware layer provides the actual computing, network and storage resources and capability. The hypervisor or the virtual machine monitor VMM provides the capability to create multiple VMs that are sharing and utilizing the hardware resources.

The virtualized computing infrastructure is created by installing a Virtual Machine Manager (VMM) on the physical hardware [59]. Virtual machines (VMs) are virtual instances managed by VMMs and isolated from each other. The VMM provides the necessary isolation and security between the multiple virtual machines running in parallel on a single physical computer (single physical server). The location where the physical servers are installed and maintained is called a datacenter (e.g. Amazon has multiple datacenters in Singapore, US Virginia, and Australia). Most widely adopted virtualiza-

tion 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 [70] 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.

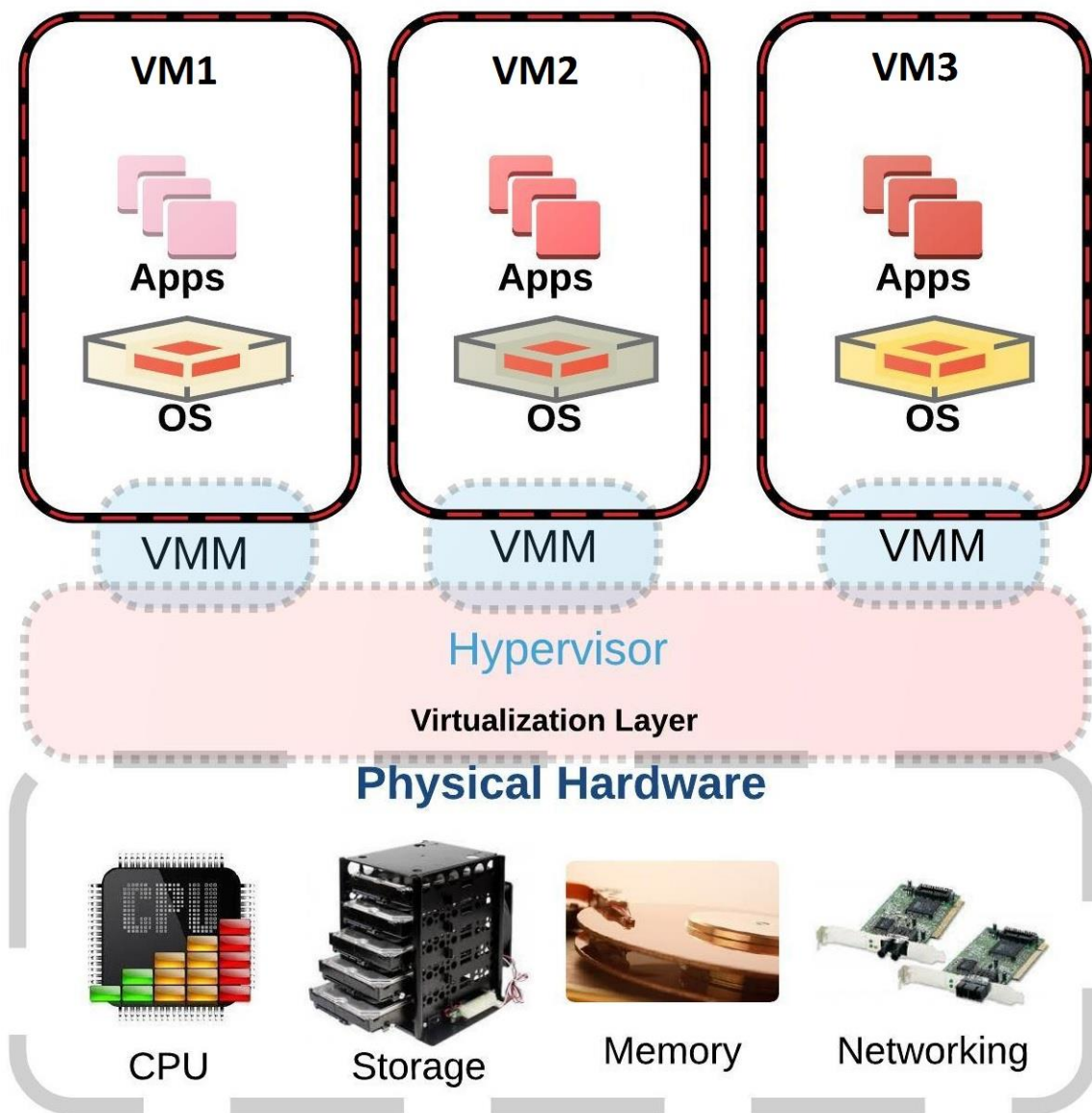


Figure 2.1: Cloud architecture and virtualization.

The following subsections discuss the key advantages of using virtualization in cloud environment [78]:

2.2.1. Resources Sharing

Resource sharing is when multiple users take advantage of the same resource to share its features. When a resource is not fully utilized by a single user, then it is better to share it among multiple users. For example, in figure 2.1, multiple VMs are created for multiple users and all are sharing the same underlying physical resources such as CPU, memory and storage.

2.2.2. Resource Isolation

Resource isolation is when the performance of one resource does not affect another resource [19]. When there is sharing of resources among users, isolation of resources is required. In figure 2.1, VM1, VM2, and VM3 are all sharing the same underlying physical resources but may belong to different users. Therefore, each VM has to be isolated from being interfered with by non-authorized users. Users using one virtual component should not be able to interfere with the activities of other users' components. This is also applicable even if different users belong to the same organization since different departments of the organization may have various levels of data confidentiality.

2.2.3. Resources Aggregation

Resources aggregation is to form a cluster of resources to reduce their load. If the resource is too small, it is possible to construct a large virtual resource by combining small resources. The combined resource may act as a single large resource instance. For example, a large storage service can be made up using many small and inexpensive storage resources.

2.2.4. Dynamics of Resources

A dynamic resource is a resource which is elastic during run-time and can adapt to system changes and requirements. Virtual resources are much easier than using physical resources where relocating is required to meet user requirements. For instance, relocating VMs across datacenters is one feature that is very challenging to apply to physical resources.

2.2.5. Ease of Resource management

Resource management is the continuous maintaining of its performance condition. However, physical resources are more difficult than virtualized resources to manage. For example, it will be much more difficult and time-consuming to configure 10 physical machines than configuring 10 virtual instances within a single physical machine.

2.3. Cloud Application Deployment

This section presents the phases of application deployment on clouds.

The application deployment determines how, when, and which provisioning operations should be processed and applied on cloud resources. A typical cloud based application (e.g. CDN applications) is multi-layered and consists of several components such as load balancers, web servers, streaming servers, application servers, and database systems [136]. Notably, each component may instantiate multiple software resources across the cloud layers as needed and when required. Such multiple instantiations can be allocated to one or more hardware resources. Technically, the aforementioned application components are distributed across cloud layers, so a number of provisioning operations

take place at design time as well as run-time. These provisioning operations should ensure SLA compliance by achieving the applications components' QoS targets. Figure 2.2, presents the application deployment in a cloud environment.

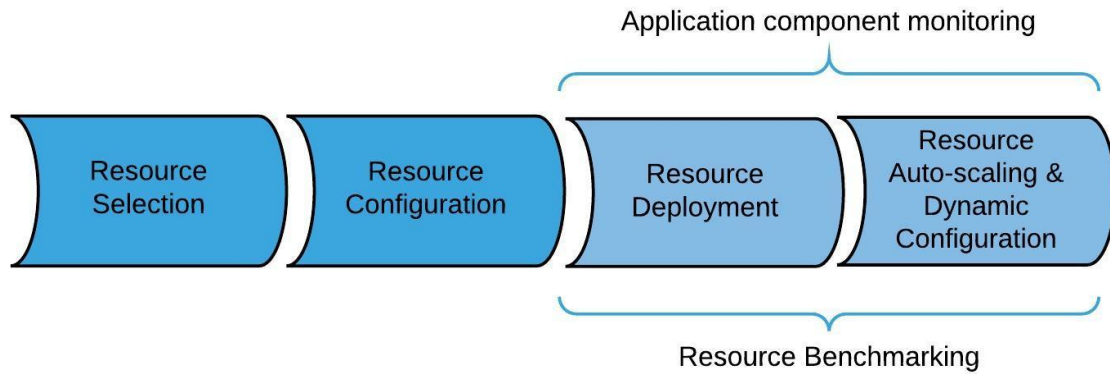


Figure 2.2: Cloud application deployment in cloud environment.

2.3.1. Resource Selection

Resource selection within the application deployment refers to the process where the system administrator selects application components (web server, multimedia server, database server, etc.) and hardware resources (CPU, storage, and network). This process encapsulates the allocation of hardware resources to those selected application components. Many resource selection algorithms have been developed and adapted for cloud computing from grid computing [121]. The resource selection processes use workload patterns to estimate and predict the resource availability[121].

Similarly, algorithms were optimized for the problem of selecting resources such as a host for the execution of cloud-based application [89]. Such optimized algorithms increase the locality of a host; that is, selecting the host with minimum propagation delay (short distance). In addition, in data-intensive environments, besides computational re-

sources, resources to be selected include data resources selection, which is equivalent to replica selection in data grid [30]. Also, a framework in [15] was developed to cluster the virtual machines to enable High Performance Computing (HPC) applications. This framework considers the compute power of VMs and the bandwidth available between VMs dynamically and selects the best set of VMs for the execution of HPC applications. The framework is based on a well-known K-means data clustering algorithm to group VMs.

2.3.2. Resource Deployment

During the resource deployment process, a system administrator instantiates the selected application components on the respective hardware resources, as well as configuring these resources for successful communication and inter-operation with the other application components already running in the system.

In cloud environments, applications require guaranteeing various SLA objectives to satisfy their QoS targets. Besides, resource utilization is of dominant importance to the cloud providers [56]. For instance, some utilization algorithms can be applied to improve resources utilization (e.g. optimized algorithm for task scheduling based on ABC (Activity Based Costing) [31]. Such an optimized algorithm considers the cost as the only SLA objective for scheduling tasks in a cloud environment. In other scenarios, the dominant factor when deploying resources is profit. In [90], two algorithms are formulated whereby the first one explicitly takes into account not only the profit achievable from the current service, but also the profit from other services being processed at the same resource instance. The second algorithm attempts to minimize the cost of renting resources from other infrastructure vendors. Other mechanisms of resource deployment

consider computational cost and data transmission cost [128]. Mainly, such approaches try to minimize the total execution cost of applications on cloud resources.

2.3.3. Resource Management

A resource is any physical or virtual component of limited availability within a computer system. Resource management in a cloud environment is the technique for managing resources, which can be controlling the access to this resource or releasing this resource [5, 175]. Further, resource management in a cloud environment needs complete provisioning, which includes resource selection and deployment. Besides, a resource provider needs to be fully aware of the resources' performing condition to apply any required auto-scaling or configuration such as adding cloud services when needed, and reducing them during the periods of low demand. Gaining knowledge of the resource status requires continuous monitoring and benchmarking which is the main focus of this thesis and will be described in the next sections in more detail.

2.3.4. Resource monitoring and benchmarking

Monitoring and benchmarking processes are applied to gain the required knowledge of the application components' performance during provisioning. This is to ensure the required performance of deployed applications (based on QoS performance parameters defined in SLA) and to avoid unwanted performing status. Moreover, cloud platform resources need to be benchmarked prior to deploying applications to test certain QoS parameters (e.g. availability, network bandwidth). Being aware of QoS will enable applications' administrators to apply any required re-configuration and auto-scaling at the right time.

2.3.5. Resource auto-scaling

Resource auto-scaling is modifying the resource capacity to ensure the required performing condition. One of the key features of a cloud is elasticity where resources meet the user's needs dynamically. This cannot be applied without the right and accurate auto-scaling process, which can be increasing or decreasing capabilities of certain cloud resources to meet the QoS targets in an SLA; for example, increasing the number of VMs or decreasing this number according to the user's requirements. To perform effective auto-scaling, it is important to understand the performance of each individual application component deployed across the cloud layers.

Nevertheless, the current monitoring and benchmarking tools are not capable of monitoring a specific component of an application (e.g. database component). Therefore, auto-scaling may not take place for a specific application component; rather, it is only based on the whole application or resource performing condition.

2.4. Cloud Resource Provisioning

Cloud resource provisioning is a complex task [29] and is referred to as the process of application deployment and management on cloud infrastructure. Current cloud providers such as Amazon EC2, ElasticHosts⁶, GoGrid⁷, TerraMark⁸, and Flexiant⁹, do not completely offer support for the automatic deployment and configuration of application

⁶ www.elastichosts.com

⁷ www.gogrid.com

⁸ www.terramark.co.nz

⁹ www.flexiant.com

components [83]. Therefore, several companies, e.g. RightScale¹⁰ and Scalr¹¹ provide scalable managed services on top of cloud infrastructures to cloud consumers to support automatic application deployment and configuration control [83].

In order to ensure the provisioning goals are met at all times, a provisioning mechanism runs continuously. The following are design goals for any provisioning approach [174]:

Automation - All decisions related to provisioning should be made automatically without the need for human intervention;

Adaptation - The application provisioner should adapt to workload uncertainties;

Performance assurance - The resource allocation in the system can be dynamically varied for satisfying achievement of QoS targets.

Furthermore, the three main steps for cloud provisioning are virtual machine provisioning, resource provisioning, and application provisioning [138, 139, 174].

2.4.1. Resource Provisioning

Resource provisioning is allocating a specific cloud resource to an application or application's component [74]. For instance, the virtual machines provisioning process refers to the process of creating VM instances on a cloud provider's underlying physical resources that match the critical characteristics (e.g. storage, memory), configurations

¹⁰ www.rightscale.com

¹¹ www.scalr.com

(software environment), and other requirements (availability zone) [54]. For illustration, Bitnami¹² enables consumers to provision a Bitnami stack that consists of VM and appliances, which are completely configured and ready to use on any cloud platform [24]. On the other hand, consumers of Amazon EC2¹³ will first require the provision of a VM on the cloud then may opt for the necessary appliances on the VM [24]. In [181], a proposed mechanism named CA-PROVISION handles the set of available computing resources as ‘liquid’ resources which, can be configured into various types of VM instances based on cloud customers’ requests. The CA-PROVISION mechanism regulates the allocation based on the users’ valuations until all resources are allocated. Also, it involves a reserve price indicated by the operating cost of the resources. Similarly, Shirako and NIMO are two systems that complement each other to obtain demand provisioning of VM instances mainly for database applications [158]. Similarly, Shirako controls the VM provisioning process while NIMO guides Shirako through active learning models [68]. Another example of the VM instances provisioning mechanism is the Business Process Execution Language (BPEL) system proposed in [52]. Primarily, PBEL is on-demand resource provisioning which supports scientific workflow by dynamically provisioning VMs in Amazon EC2.

2.4.2. Application Provisioning

Application provisioning is the process of application deployment on VMs on the cloud infrastructure. For example, deploying a Tomcat¹⁴ server as an application component on a VM hosted on the Amazon EC2 cloud. Applications provisioning can be done in two ways. The first method consists of deploying the application components together while hosting a VM. In the second method, the consumer may want to first deploy the

¹² <https://bitnami.com>

¹³ <https://aws.amazon.com/ec2/>

¹⁴ tomcat.apache.org/

VM, and then as a separate step, the consumer may deploy the applications. Figure 2.3, presents a sequence diagram to illustrate the applications and resources provisioning.

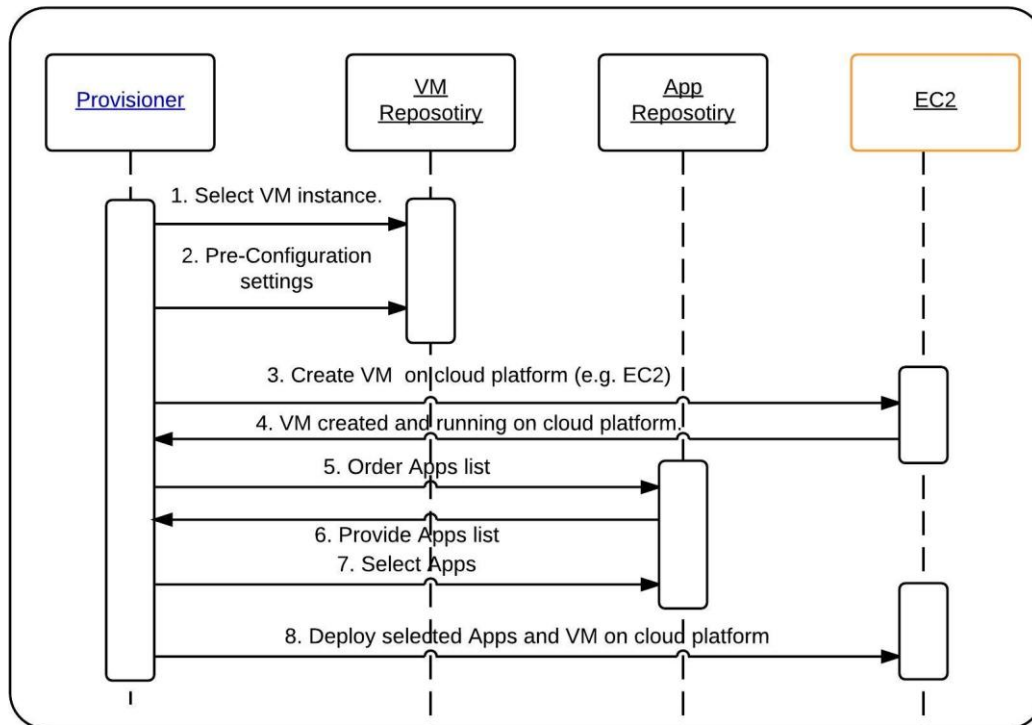


Figure 2.3: Provisioning and deployment sequence diagram

- *Step 1* - from the VM repository, a consumer views the available VMs provided by the cloud platform and selects the preferable VM instance type.
- *Step 2* - the consumer sets up his/her preferences/configurations on this VM.
- *Steps 3 and 4* - the user deploys this VM on the cloud platform successfully. Subsequently,
- *Steps 5 and 6* - the consumer retrieves back a list of available applications from the applications repository.

- *Step 7* - the consumer simply opts for his/her desired application components that he/she would like to provision.
- *Step 8* - the cloud consumer deploys the application components and the VM on the cloud platform.

After the provisioning stage, a cloud workflow instance might be composed of multiple cloud resources, and in some cases resources from a number of different resources providers. Therefore, monitoring the QoS performance of cloud applications' on multi-clouds become much more complex [101]. Furthermore, at run time, the QoS of the running instance needs to be consistently monitored to guarantee the SLA and avoid/handle abnormal system behavior. Monitoring is the process of observing and tracking applications/resources at run time. It is the basis of control operations and corrective actions for running systems on clouds. Despite the existence of many commercial monitoring tools in the market, managing service level agreements (SLAs) between multiple cloud providers still pose a major issue in clouds.

In some way, cloud monitoring, benchmarking, SLA and dynamic configuration are correlated in the sense that one has an impact on another. In other words, enhancing monitoring and benchmarking functionalities will in turn assist meeting SLAs as well as improving dynamic configuration operations at run time. Moreover, an SLA has to be met by the cloud providers in order to reach the required reliability level required by consumers. Also, auto-scaling and dynamic configurations are required for optimal use of cloud technology. This all-together leads us to the conclusion that the cloud monitoring and benchmarking process is a key element of cloud operations. However, the limitation with current monitoring and benchmarking frameworks (e.g. Cloudwatch, Azure

FC which do not support cross-layers and multi-clouds monitoring and benchmarking) render the need to further study and enhance the cloud monitoring and benchmarking process.

2.5. Applications Management Challenges in Cloud Environment

Cloud resource management as presented in section 2.3.3 refers to the technique for managing the resource, which can be controlling the access to this resource or releasing this resource. In general, a cloud resource, which could be the application's component container in cloud platform, needs continuous management. Furthermore, resource management may involve a wide range of different scenarios. In parallel to these scenarios, management challenges may arise at any point in time.

2.5.1. Resource Scenarios in Cloud environment

The following examples characterize some of the different resource management situations that can take place in cloud environments [43]:

2.5.1.1 Task submission

Task submission is when a resource accepts responsibility to perform a specified task (e.g. execute a program, move a file, or perform a database function). Ideally, this represents a basic type of resource management between a resource provider and a consumer in which a resource provider commits to perform the agreed function with the resource consumer.

2.5.1.2 Workload management

Workload management refers to the extension of task submission described earlier by provisioning tasks to provide a specified level of capability (such as processors on a computer, threads or memory in a server, bandwidth on a network, or disk space on a storage system). This would enable the application manager to gain control over the aspects of how such a task is performed over time.

2.5.1.3 Advanced reservations

In which a resource capability becomes available by its local manager to other users to be reserved for use [167]. This type of resource management can be particularly important for heterogeneous resource sharing where each resource is owned by a different institution [131].

2.5.1.4 Co-scheduling

Co-scheduling refers to resources when they are made to be available simultaneously by organizing advanced reservation agreements across the required resources. This type of management function is characterized by data transfer services in which data source and storage systems must be coordinated along with network bandwidth. In this condition, multiple compute resources should be available at the same period of time [44, 61].

For the aforementioned resource scenarios, resource providers need to apply resource management in the form of monitoring, benchmarking, and control, which is discussed in the following sections.

2.5.2. Resource Monitoring

A continuous monitoring process is desirable to ensure that the deployed software and hardware resources run at the required level of performance to satisfy the SLA. This process involves detecting and gathering information about the running resources. In case of the detection of any abnormal resource behavior, the resource administrator is notified for policy-based corrective actions to be undertaken as a resource remedy. Section 2.6 will discuss further the cross-layers and multi-clouds monitoring process.

2.5.3. Resource Benchmarking

Conceptually, benchmarking is a quantitative foundation of computer system and architecture research. Benchmarks are used to experimentally determine the benefits of new designs [104]. Furthermore, benchmarking is a standard whereby a cloud resource can be tested to determine its performance compared to its peers. Moreover, prerequisite metrics and indicators for cloud resource have to be identified for the benchmarking purposes. Section 2.7 will discuss further the cross-layers and multi-clouds benchmarking process.

The next section explores and discusses the features of commercial and open-source monitoring and benchmarking frameworks.

2.6. Cloud Monitoring

2.6.1. Monitoring Process

In clouds, monitoring is essential to maintain high system availability and performance of a certain resource and it is important for both resource providers and consumers [47,

69, 116]. Primarily, monitoring is a key tool for (i) managing software and hardware resources, and (ii) providing continuous information for those resources as well as for consumer hosted applications on the cloud. Cloud activities like resource planning, resource management, datacenter management, SLA management, billing, troubleshooting, performance management, and security management essentially need monitoring for effective and smooth operations of all the system's resources [2]. Consequently, there is a strong need for monitoring, looking at the elastic nature of cloud computing [157]. In cloud computing, monitoring can be of two types: high-level and low-level. High-level monitoring is related to the virtual platform status [33]. Low-level monitoring is related to information collected about the status of the underlying physical infrastructure [33, 163]. The cloud monitoring framework is a self-adjusting and typically multi-threaded framework that is able to support monitoring functionalities [11]. It comprehensively monitors pre-identified instances/resources on the cloud for abnormalities. On detecting an abnormal behavior, the monitoring framework attempts to auto-repair this instance/resource if the corresponding monitor has a tagged auto-heal action [11]. In case of auto-repair failure or an absence of an auto-heal action, a support team is notified. Technically, notifications can be sent by different means such as email, or SMS [11].

2.6.2. Monitoring, QoS, and SLA

As mentioned above, cloud monitoring is needed for continuous assessment of resources or application components on cloud environments in terms of performance, reliability, power usage, ability to meet SLA, security, etc [87]. Fundamentally, the monitoring process can be computation based and/or network based. The computation based monitoring process is concerned with the status of the real or virtualized platforms and infrastructure running cloud application components. Data metrics considered in such a process may include CPU speed, CPU utilization, disk throughput, VM acquisi-

tion/release time and system up-time. Network based measurements focus on network layer data related metrics like jitter, round-trip time RTT, packets loss, traffic volume etc. [166].

At run-time, a set of operations takes place in order to meet the QoS parameters specified in an SLA document that guarantees the required performance objectives of the cloud consumers. Being aware of the system's current application components and hardware resources status is imperative for handling such uncertainties to ensure the fulfillment of QoS targets [155]. In addition, detecting exceptions and malfunctions while deploying application components on hardware resources is essential e.g. showing QoS delivered by each application component (software resource such as web server or database server) hosted on each hardware resource. Uncertainties can be tackled through the development of an effective, scalable, interoperable monitoring framework with easy-to-use interfaces.

Infrastructure providers try to utilize idle resources by renting them to other users in order to gain profit. Hence, providers accept new requests aiming to increase the profit. However, they must guarantee QoS targets based on the settled SLA with their customers. In this case, the SLA is acting as an association between both the service provider and the service customer. For this purpose, an SLA summarizes a number of metrics, on the basis of which users can specify their requirements. Characteristically, QoS targets in cloud environment vary across application types. For example, QoS targets for eResearch applications are different from static, single-tier web applications (e.g. web site serving static contents) or multi-tier applications (e.g., on demand audio/video streaming). Based on application types, there is always a need to negotiate different SLAs.

Hence, the SLA document includes conditions and constraints that match the nature of QoS requirements with each application type. For example, a bio-informatics applications running a genome analysis experiment on cloud resources will only care about data transfer (upload and download) network latency and processing latency. On the other hand, for audio/video streaming applications, the quality of the transferred data over network is more important. Hence, other parameters gain priority in certain situations. Failing to track QoS parameters will eventually lead to SLA violations. Consequently, monitoring is fundamental and responsible for an SLA's compliance certification [108]. Moreover, the cross-layers application monitoring approach can provide significant insights into the application performance to both the consumer and cloud provider. This is essential for consumers as they can identify and isolate application performance bottlenecks in specific layers. From a cloud provider point-of-view, the QoS statistics on application performance across-layers can help them maintain their SLAs, delivering better performance and higher consumer satisfaction.

2.6.3. Monitoring across Different applications, Cross-Layers, and Multi-Clouds

As mentioned previously, application components (e.g., streaming server, web server, indexing server, compute service, storage service, and network) are distributed across cloud layers including PaaS and IaaS. Thus, in order to guarantee the achievement of QoS targets for the application as a whole, monitoring QoS parameters should be performed across all layers of the cloud stack including Platform-as-a-Service (PaaS) (e.g., web server, streaming server, indexing server, etc.) and Infrastructure-as-a-Service (IaaS) (e.g., compute resources, storage resources , and network). Figure 2.4 illustrates how different components in a cloud platform are distributed across the cloud platform layers. Table 2.1 shows the QoS parameters that a monitoring tool should consider at each cloud layer.

The current cloud-application monitoring frameworks such as Amazon CloudWatch¹⁵ typically monitors the entire VM as a black box. This means that the actual behavior of each application's component is not monitored separately. This renders application monitoring with a limited scope where not all components distributed across PaaS and IaaS layers are monitored and benchmarked holistically. This limiting factor reduces the ability for fine-grained application monitoring and QoS control across-layers. Moreover, current cloud monitoring frameworks are mostly incompatible across multiple cloud providers. For example, Amazon CloudWatch does not allow monitoring of application components hosted on non-AWS platforms. Furthermore, Windows Azure Fabric Controller does not enable multi-clouds monitoring. This defeats the distributed nature of cloud application hosting. These drawbacks trigger the significance of having interoperable and cross-layer multi-clouds enabled monitoring techniques and frameworks.

¹⁵ <http://aws.amazon.com/cloudwatch/>

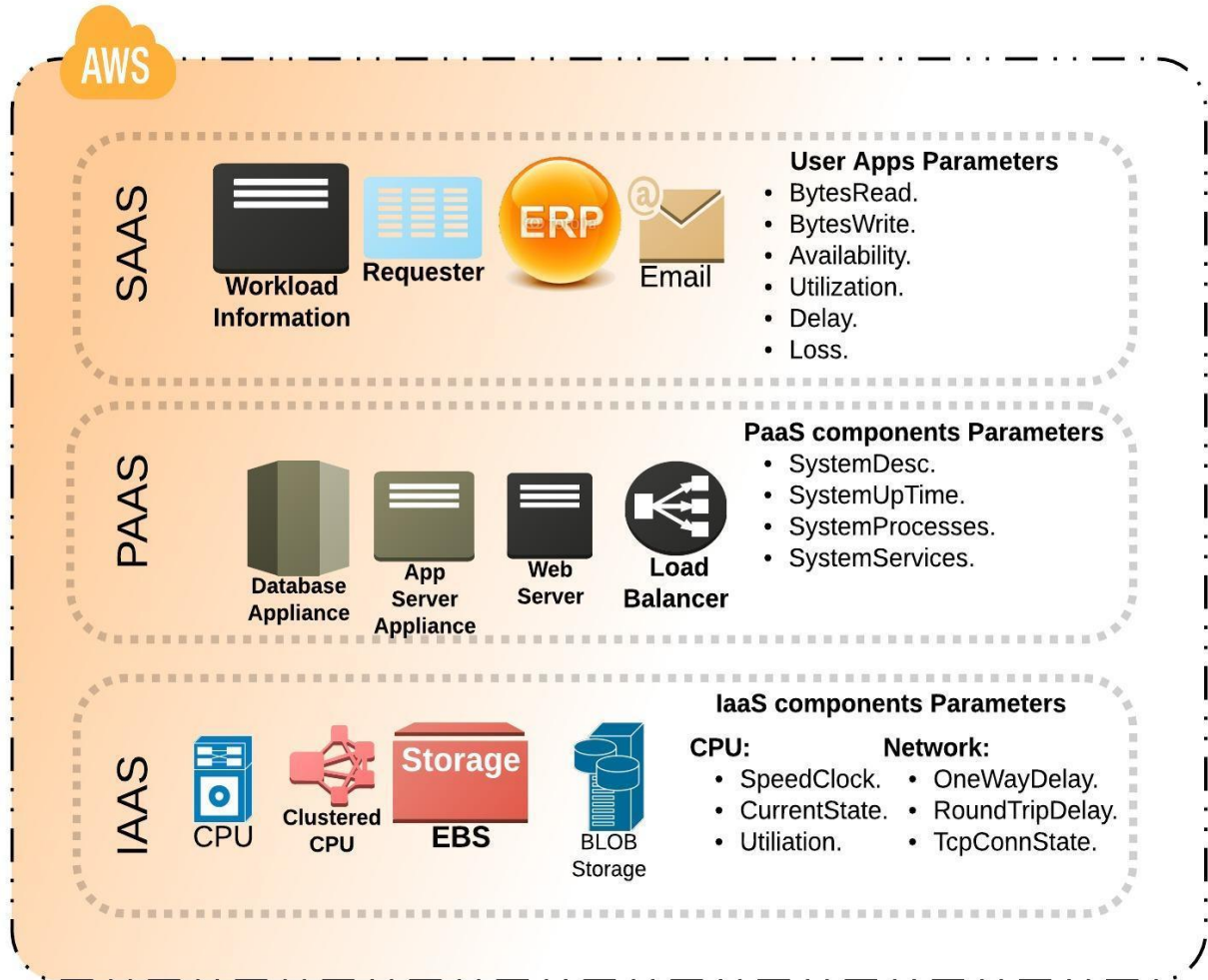


Figure 2.4: Components across cloud platform layers.

Cloud Layer	Layer Components	Targeted QoS Parameters
SaaS	Appliances x,y,z, etc.	BytesRead, BytesWrite, Delay, Loss, Availability, Utilization.
PaaS	Web Server, Streaming Server, Index Server, Apps Server, etc.	BytesRead, BytesWrite, SysUpTime, SysDesc, HrSystemMaxProcesses, HrSystemProcesses, SysServices.
IaaS	Compute Service, Storage Service, Network, etc.	CPU Parameters: (Utilization, ClockSpeed, CurrentState).

Table 2-1: QoS parameters at each cloud platform layer.

2.7. Cloud Benchmarking

2.7.1. Benchmarking Definition

The diversity of cloud providers leads to a practical question: *how well does a cloud provider perform compared to the other providers?* Answering this question will benefit both cloud customers and providers. For a potential customer, the answer can help choose a provider that best fits their performance and cost needs. For instance, a customer may choose one provider for storage intensive applications and another for computation intensive applications. For a cloud provider, such answers can point to the right directions for improvements. For instance, a provider should pour more resources into optimizing table storage if the performance of its store lags behind competitors.

A brief definition for benchmarking can be as following “A benchmark is a program which generates a well-known workload on the system under test and enables the expert to measure a set of predefined performance indexes” [135].

2.7.2. Benchmarking, QoS, and SLA

In a cloud computing environment, the QoS parameter values are stochastic and can vary significantly based on unpredictable user workloads, hardware and software failures, thereby necessitating the awareness of system’s current software and hardware service status such that QoS targets of cloud-hosted applications are met. Hence, besides cloud monitoring, benchmarking in cloud can assist in the holistic monitoring and awareness of applications’ components at *aaS layers to meet SLAs. Additionally, benchmarking can be used for: (i) understanding application components’ performance (resource and network) before application deployment; (ii) facilitating application base lining; and (iii) enabling continual comparison of applications’ QoS performance against baseline results. Recently, both industry and academia have focused on cloud monitoring and benchmarking [16]. However, most of the approaches are limited to one cloud provider and/or one cloud platform layer (IaaS/PaaS/SaaS).

In addition, there are several benchmarks for evaluating High Performance Computing (HPC) servers, web servers, and database servers. However, these traditional benchmarks are not essentially appropriate for evaluating cloud platform resources because of the differences between a cloud environment and traditional computer systems [77].

2.7.3. Cross-Layers Benchmarking on Multi-Clouds

While there has been significant interest in the area of cloud monitoring by academia and industry, most of the existing approaches [94, 185] suffer from several problems; for example, they are tightly coupled to a particular cloud platform and can only perform monitoring or benchmarking at a particular cloud layer (e.g., either IaaS or PaaS or SaaS). There is no existing benchmark suite for evaluating cloud performance on the whole system level. Furthermore, many other benchmarking efforts have been done from the industry and research, like VMware VMmark, Intel vConsolidate. But, these projects only address high-level performance's characterization of co-located virtual machines [55].

Therefore, it is asserted that in a distributed application hosting environments such as clouds, there is a need for application deployment across multi-cloud providers and multi-layered environments to benefit from security, QoS, resiliency, availability and economies of scale. For example, in terms of QoS, different public clouds may perform differently. For instance, the recent results presented by Leitner and Cito [92] show that Amazon EC2 and Google App Engine may perform differently with similar cloud configurations, regions and cost. Thus, it may make sense to use a particular cloud provider over another provider given the type of VM configuration and region requirements. Further, in terms of availability and resiliency, it is also suggested that multiple cloud providers are supported for deploying applications [92].

Additionally, a study made over four different cloud providers show that these providers vary significantly in terms of performance and cost [96]. For illustration, this study shows the following:

- I. Cloud instances are not equally cost-effective. For example, while only 30% more expensive, a fourth provider's virtual instance can be twice as fast as that of first provider's.
- II. The second provider allows its virtual instance to fully utilize the underlying physical machine when there is no local resource competition. Therefore, an instance can achieve high performance at low cost.
- III. The performance of the storage service can vary significantly across the providers. For instance, the first provider's table query operation is an order of magnitude faster than that of the others.
- IV. The providers offer dramatically different intra-datacenter bandwidth, even though intra-datacenter traffic is free of charge. For instance, the first provider's bandwidth is on average three times higher than the second provider's.

This necessitates QoS monitoring and benchmarking cross-layers. For example, the failure of a particular VM (IaaS layer) affects the QoS of web application (PaaS layer) or database application (PaaS layer) hosted within that VM. This ultimately affects the QoS of the end-user of that web application offering (SaaS layer). This establishes the need for the cloud monitoring and benchmarking framework that is capable of benchmarking applications and components across multiple cloud layers and across multiple cloud provider environments. Further, benchmarking aids in ensuring that the system's current performance is as good as its baseline performance.

2.8. Evaluation Framework

This section presents the basic components that can be considered as evaluation dimensions in order to evaluate a monitoring or benchmarking framework in a cloud computing environment. This is motivated by the lack of work that evaluates monitoring or benchmarking tools.

2.8.1. Monitoring and Benchmarking Framework Architecture

A monitoring and benchmarking technique architecture refers to the technique, which is applied for communicating and processing monitoring and benchmarking related performance QoS parameters. Technically, it is a way of connecting a monitoring and benchmarking technique with an application's components and cloud resources to collect related statistical data. Typically, network monitoring and benchmarking can be performed on centralized and de-centralized network architectures as shown in figure 2.5.

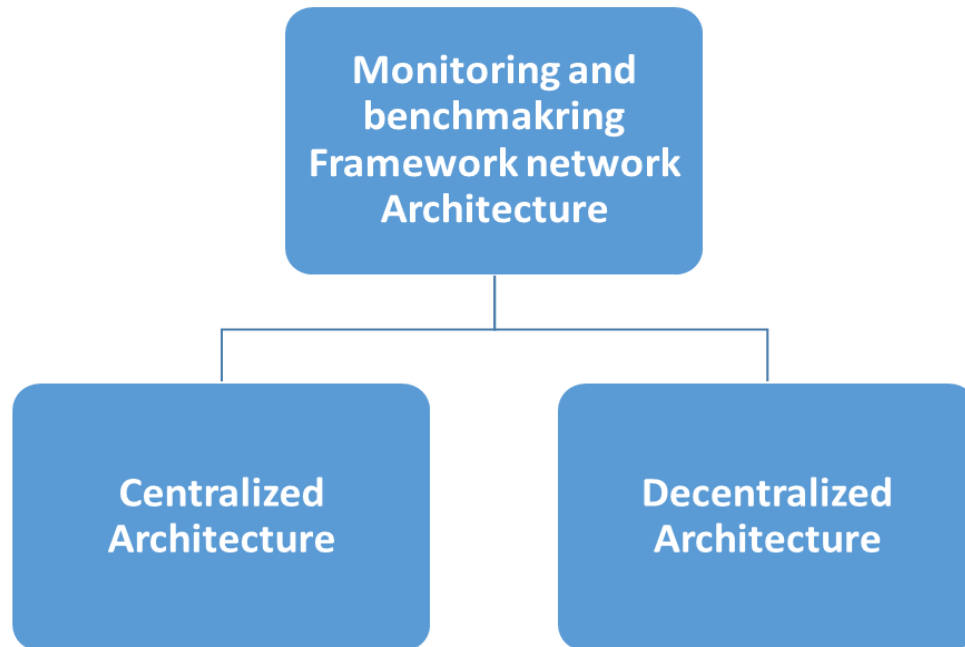


Figure 2.5: Framework network architecture.

2.8.1.1 Centralized Architecture

In the centralized architecture shown in figure 2.6, the PaaS and IaaS resources send QoS status update queries to the centralized monitoring/benchmarking server. In this scheme, the monitoring and benchmarking techniques continuously pull the information from the components via periodic probing messages. In [11], the authors show that a centralized architecture allows better management for cloud applications.

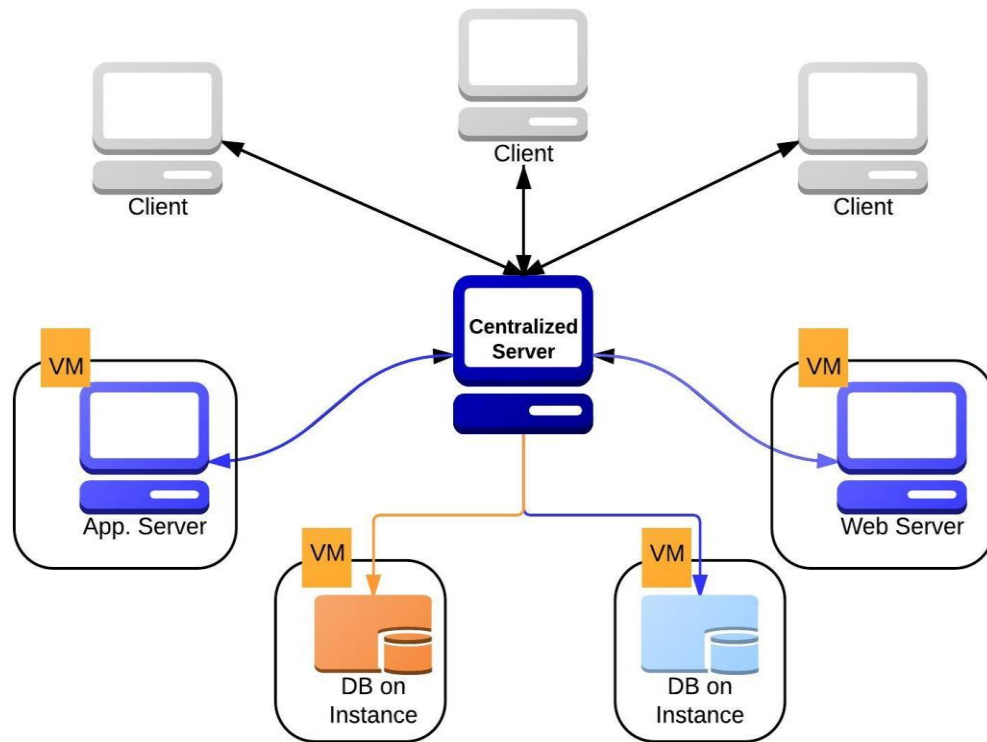


Figure 2.6: Centralized monitoring/benchmarking framework architecture.

In Mobile Cloud Computing (MCC), an increased security level for mobile devices is achieved by a centralized monitoring and maintenance of software [84].

Nevertheless, a centralized approach has several design problems, including:

- Being prone to a single point of failure;
- Lack of scalability;
- High network communication cost at links leading to the information server (i.e., network bottleneck, congestion); and
- A possible lack of the required computational power to serve a large number of monitoring requests.

2.8.1.2 Decentralized Architecture

Peer-to-peer networking development have become of interest to both business and research communities. Today, many peer-to-peer frameworks have been developed to provide overlay infrastructures to create applications such as dynamic file sharing, content distribution, application multicast, VoIP services, and DNS systems [188]. Further, some protocols of peer-to-peer started to enhance the overall performance of such overlay nodes [88, 177].

New proposals for decentralized cloud monitoring and benchmarking tools have gained momentum. Figure 2.7 shows the broad schematic design of a decentralized cloud monitoring/benchmarking framework. The decentralization of monitoring/benchmarking framework can overcome the issues related to current centralized systems. A monitoring/benchmarking framework configuration is considered as decentralized if none of the components in the framework is more important than the others. In case one of the components fails, it does not influence the operations of any other component in the framework.

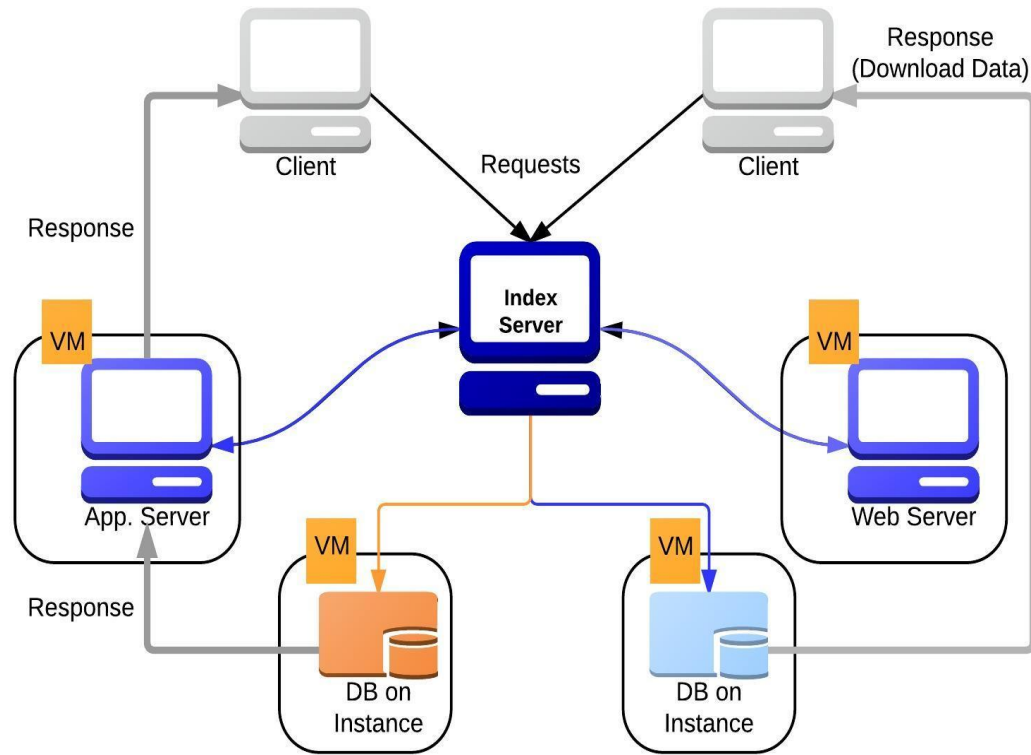


Figure 2.7: Decentralized monitoring/benchmarking framework architecture.

- Structured peer-to-peer Overlay

Looking forward, to have a network layout where a central authority is disabled has led to the development of the structured peer-to-peer networks. In such a network overlay, a central point of failure is eliminated [45, 46]. Often, structured peer-to-peer frameworks are denoted as Distributed Hash Tables (DHTs) [21]. In DHTs, a variant of regular hashing is used to allocate the ownership of each file to an individual peer. This enables peers to search for resources on the network using a hash table, which is, (key, value) pairs that are stored in the DHT, and any joining node can efficiently regain

the value associated with a given key [34]. The basic service of such protocols is named Key-based Routing service (KBR). Further, such protocols are different in many characteristics (e.g. overlay topology and routing table maintenance). Nevertheless, to route traffic proficiently through the network, nodes in a structured overlay must continue maintaining lists of neighbors that meet specific criteria [97]. Therefore, they are less robust in networks with large numbers of nodes frequently joining and leaving the network. Examples of structured peer-to-peer networks include (e.g. CAN [140], Chord [165], Pastry [147], and Tapestry [186]).

- Unstructured peer-to-peer Overlay

Unstructured peer-to-peer networks overlay is meant to be a distributed overlay but the difference is that the search directory is not centralized unlike structured peer-to-peer networks overlay, which leads to absolute single point failure in such network overlay [63]. In other words, the form of connections between nodes are random. Since there is no compulsory structure applied globally on such networks overlay, they are considered easier to build and allow for localized optimizations to different regions of the overlay [35]. Well known examples of unstructured peer-to-peer networks include (e.g. Napster [159], and Gnutella [144]). Unlike structured peer-to-peer overlay, unstructured peer-to-peer overlay are considered to be more robust because the role of all peers in the network is the same.

- Hybrid peer-to-peer Overlay

A Hybrid peer-to-peer network system is a combination of structured and unstructured peer-to-peer network systems. Super peers can act as local search hubs in small portions of the network whereas the general scope of the network behaves as unstruc-

tured peer-to-peer system [91]. A typical model of a Hybrid peer-to-peer network is having a centralized peer acting as a server which, helps peers find each other. Kazaa is a hybrid of centralized Napster and decentralized Gnutella network systems [100]. Technically, Hybrid peer-to-peer network systems provide better performance over pure structured networks and unstructured networks overlay since some specific functions involve a centralized functionality and at the same time get advantage from the decentralized aggregation of peers provided by unstructured networks [178].

2.8.2. Interoperability

The interoperability perspective in technology focuses on the system's technical capabilities to interface between organizations and systems. It also focuses on the resulting mission of compatibility or incompatibility between systems and data collation partners. Modern business applications developed on cloud are often complex and require interoperability capability. For example, an application owner can deploy a web server on Amazon cloud while the database server may be hosted in Azure cloud. Unless data and applications are integrated across multiple clouds properly, the results and benefits of cloud adoption cannot be achieved. Furthermore, interoperability is also necessary to avoid cloud provider lock-in. In order to achieve cooperation and federation, companies and organizations' cloud services must be able to effortlessly interact with different and heterogeneous PaaS services. However, studies on existing cloud platforms show that such PaaS services require the use of specific and proprietary APIs [154]. For example, to interact with the Force.com PaaS Apex, REST API is provided, Cloud Foundry¹⁶ is delivered with a proprietary API (i.e. the Cloud Foundry core REST API), etc [151]. Further, each existing PaaS exposes a different interface and no standardized API is reachable by

¹⁶ <http://www.cloudfoundry.com/>

PaaS resources consumers. Thus, the actual PaaS provider's policy makes a seamless interaction with their PaaS a very difficult task.

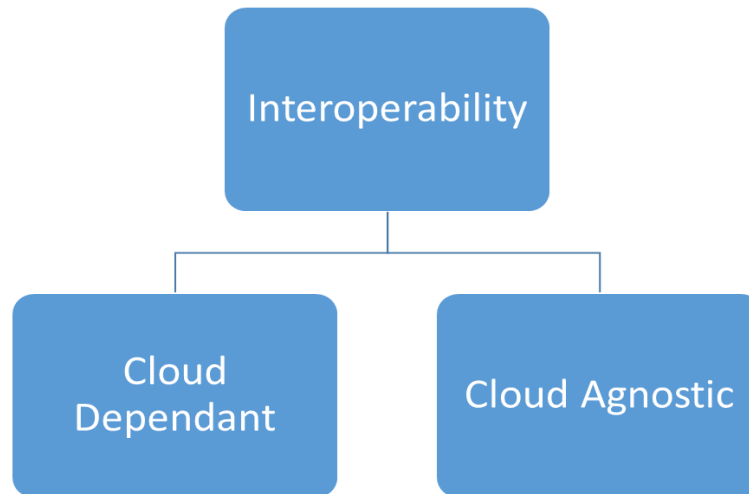


Figure 2.8: Interoperability classification.

For the above issues and requirements, this dimension (Interoperability) refers to the ability of a cloud monitoring framework to monitor and benchmark application components that may be deployed on multi-clouds. While it is not difficult to implement a cloud-specific monitoring/benchmarking framework, to design a generic cloud monitoring/benchmarking framework that can work on multi-clouds remains a challenging problem. Next, the interoperability (Figure 2.8) of monitoring/benchmarking frameworks is classified into the following categories: Cloud Dependent and Cloud Agnostic.

2.8.2.1 Cloud Dependent

Currently many public cloud providers provide their consumers monitoring/benchmarking tools to monitor or benchmark their application's CPU, storage and

network usage. Often these tools are tightly integrated with the cloud provider's existing tools. For example, CloudWatch [40], offered by Amazon is a monitoring tool that enables consumers to manage and monitor their applications residing on AWS EC2 (CPU) resources. But, this monitoring tool does not have the ability to monitor an application component that may reside on another cloud provider's infrastructure such as GoGrid or Microsoft Azure. Table 2.2 in section 2.10 illustrates some examples of cloud monitoring and benchmarking frameworks that are specific to a cloud provider as well as Cloud Agnostic.

2.8.2.2 Cloud Agnostic

In contrast to single cloud monitoring/benchmarking, engineering cloud agnostic monitoring/benchmarking tools is challenging. This is primarily due to the fact that there is not a common unified application programming interface (API) for calling on cloud computing services' runtime QoS statistics. Though recent developments in cloud programming API including Simple Cloud¹⁷, Delta Cloud¹⁸, JCloud¹⁹, and Dasein Cloud²⁰ simplify inter-action of services (CPU, storage, and network) that may belong to multiple clouds, they have limited or no ability to monitor run-time QoS statistics and application behaviors. In this scenario, monitoring/benchmarking tools are expected to be able to retrieve QoS data of services and applications that may be part of multiple clouds. Cloud agnostic monitoring/benchmarking tools are also needed if one wants to realize a hybrid cloud architecture involving resources from private and public clouds.

¹⁷ www.simplecloud.info

¹⁸ <https://deltacloud.apache.org>

¹⁹ <https://jclouds.apache.org>

²⁰ www.dasein.org/api/dasein-cloud

2.8.3. Quality of Service (QoS) Matrix

It is a non-trivial task for application administrators to understand what QoS parameters and targets they need to specify to monitor or benchmark across each layer of a cloud stack including PaaS (e.g., web server, streaming server, indexing server, etc.) and IaaS (e.g., compute resources, storage resources, and network). As shown in figure 2.9, this can be classified by one parameter or a group of parameters.

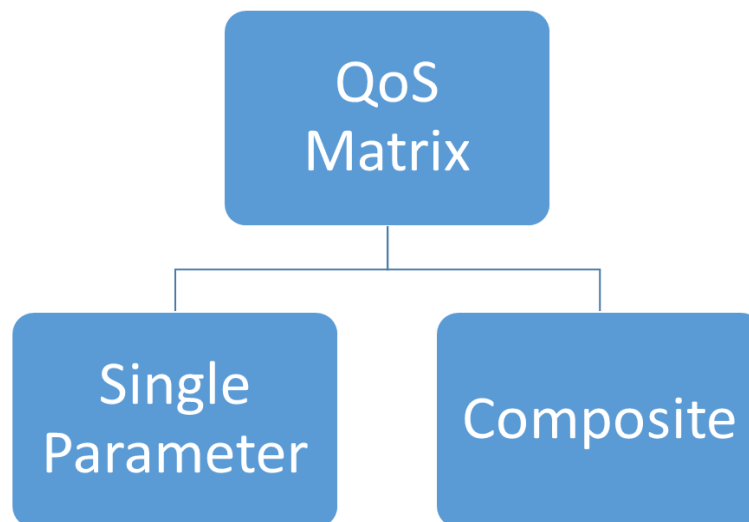


Figure 2.9: QoS matrix classification.

2.8.3.1 Single Parameter

In this scenario, a single parameter refers to a specific system QoS target. In each system, there are major atomic/single values that have to be tracked closely and continuously. For example, CPU utilization is basically expressed by only one single parameter in the SLA. Such parameters can affect the whole system and a violation in SLA can lead to a serious system failure. Unlike composite parameters where a single parameter might not

be of priority to the system administrator, single parameters in most cases gain high priority when monitoring SLA violations and QoS targets.

2.8.3.2 Composite Parameters

In a composite parameter scenario, a group of different parameters are taken into consideration. In a cloud environment, deployed application is composed of several components running on different resources. Thus, the performance quality can be determined by collective behaviors of those components [169]. After observing multiple parameters for estimating a functionality of one or more concerned processes, one result could be obtained to evaluate the QoS. To illustrate this point, “loss” parameter can be considered as a composite parameter of two single parameters: “one way loss” and “round trip loss”. Similarly, “delay” can be considered as a composite parameter of three single parameters: “one way delay”, “RTT delay”, and “delay variance”. Table 2.2 in section 2.10 shows a list of some commercial tools for cloud monitoring and benchmarking and it illustrates which of them support or do not support monitoring multiple QoS parameters.

2.8.4. Cross-Layer Monitoring and Benchmarking

As shown in figure 2.10, application components (streaming server, web server, indexing server, compute service, storage service, and network) related to an audio/video streaming application are distributed across cloud layers including PaaS and IaaS. In order to guarantee the achievement of QoS targets for such an application as a whole, it is critical to monitor or benchmark QoS parameters across multiple layers [122]. Hence, the challenge here is to develop a monitoring framework that can capture and reason about the QoS parameters of application components across IaaS and PaaS layers. As

demonstrated in figure 2.11, the visibility of commercial monitoring/benchmarking tools are classified into following categories: Layer specific and Layer Agnostic.

2.8.4.1 Layer specific

Cloud resources are distributed among three layers namely, SaaS, PaaS, and IaaS. Monitoring/benchmarking tools originally are oriented to perform monitoring/benchmarking tasks over resources only in one of the aforementioned layers. Most of present day commercial tools are designed to keep track of the performance of resources provisioned at the IaaS layer. For example, CloudWatch is not capable of monitoring information related to load, availability, and throughput of each core of CPU resources and its effect on the QoS (e.g., latency, availability, etc.) delivered by the hosted PaaS resources (e.g., J2EE application server). Hence, there exists a considerable gap and research challenges in developing a monitoring/benchmarking tool that can monitor or benchmark QoS statistics across layers of the cloud stack.

2.8.4.2 Layer Agnostic

In contrast to the previous scenario, cross-layer monitoring/benchmarking enables the consumers to gain insights to applications' components performance across multiple layers. For example, consumers can retrieve data at the same time from PaaS and IaaS for the same application. This type of cloud monitoring/benchmarking is essential in all cases but obviously it is more effective for consumers requiring complete awareness about their cloud applications.

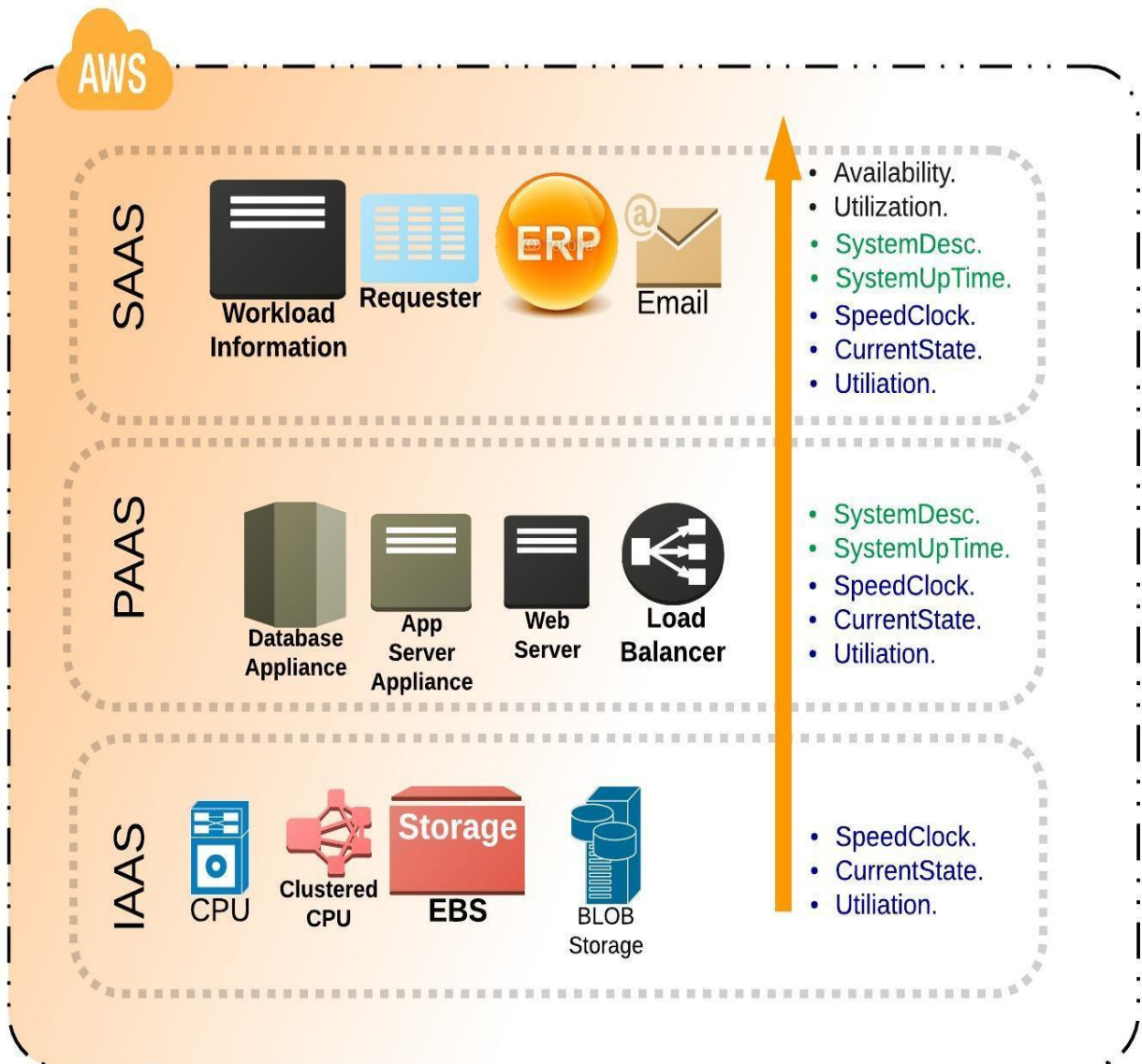


Figure 2.10: Components across cloud platform layers and QoS propagation.

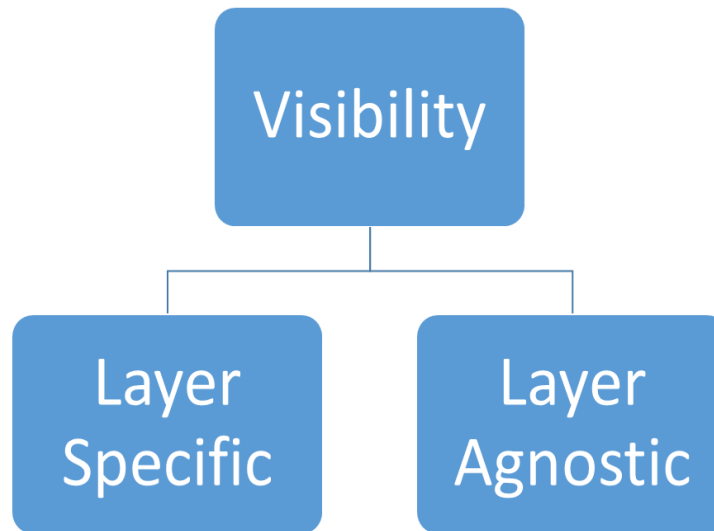


Figure 2.11: Visibility categorization.

2.8.5. Programming Interfaces and Communication Protocols

In computer programming, a programming interface is a set of routines, protocols, and tools for building software applications. A programming interface expresses a software component in terms of its operations, inputs, outputs, and underlying types (figure 2.12).

2.8.5.1 Application Programming Interface

An application programming interface (API) is a particular set of rules ('code') and specifications that software programs follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.

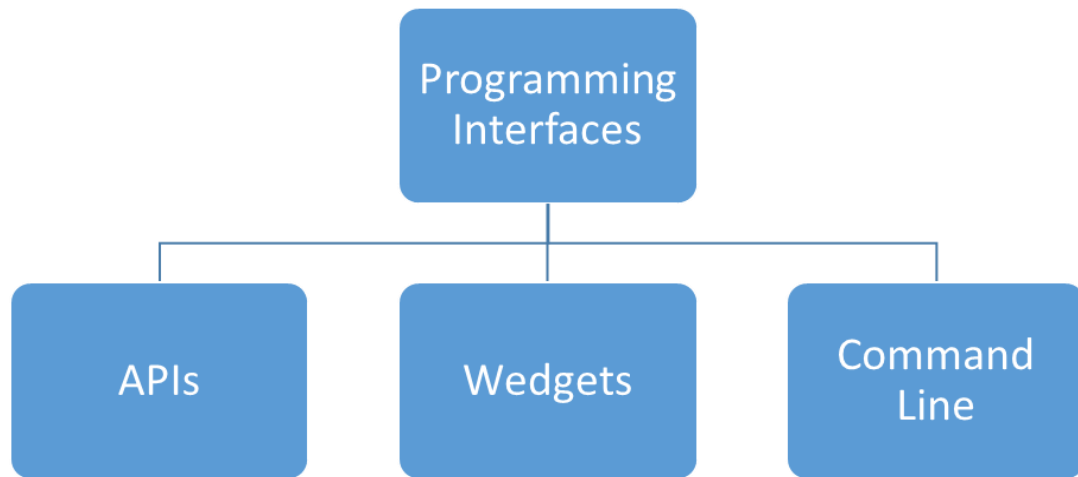


Figure 2.12: Different types of programming interfaces.

APIs allow accessing databases and/or computer hardware; in addition, APIs can facilitate the work of programming Graphical User Interface (GUI) components. For illustration, APIs ease integration of new features into existing application, which are so called plug-in APIs. Nowadays, software developers rely on pre-defined software frameworks, which are presented mainly as APIs [173]. APIs enable developers to reuse libraries, programming paradigms, and task delegation, in order to deliver useful systems rapidly and with high-quality code. Frameworks and libraries offer packaged solutions for a set of common problems in some specific domain, for example, the signal processing libraries of Matlab [27] or a GUI framework. They provide leverage in large part because they are used by many applications through a published API [148].

These days, most commercial monitoring tools such as Rackspace²¹, Nimsoft²², RevealCloud²³, and LogicMonitor²⁴ provide their consumers with extensible open APIs enabling them to specify their own required system functionalities [98].

2.8.5.2 Command-Line Interface (CLI)

Usually known as Console User Interface and Character User Interface (CUI), a command line provides a means of communication between a consumer and a computer that is based solely on textual input and output. Such inputs are converted into appropriate operating system functions. For example, in Unix, specification and parameters are processed through the command line (e.g. *argc*, *argv*, and *geropt*) [41]. Typically, command line arguments have different types (e.g. integer, string, input-file, output-file, or flag). All these arguments except the flag argument can be followed by 0 or other parameters, which are not optional in some cases. A certain value is assigned to an argument, or otherwise, a user is requested to insert a value to an argument [41]. Advanced computer users mostly prefer to use a command line as it provides them with more powerful means and control over the operating system. In addition, with CLI, scripting is a potential technique where users can save input and output of the shell commands into a text file [160]. Basically, scripts are collections of commands which the shell can read from the stored file and then process them as if they were inserted using the keyboard.

²¹ www.rackspace.com

²² <https://support.nimsoft.com>

²³ copperegg.com/tag/revealcloud

²⁴ www.logicmonitor.com

2.8.5.3 Widgets

In computer software, a widget is a software service available to consumers for running and displaying applets via a graphical interface on the desktop. Basically, widgets can be described as installed small applications with limited functionalities which can be executed within a web page. Widgets can be presented in toolkits to provide reliable and configurable interface widgets known as groupware widgets [95]. Groupware widgets include shared scrollbars, group text editors and shared canvas [146]. Users of such groupware widgets can re-position and configure in few lines of code. Moreover, groupware widgets promote group awareness and provide consistent views on shared information [95, 130]. The Toolkit in [51] presented 3D widgets, which encapsulate geometry graphs and behavior graphs. Operations are performed on such 3D widgets through a high-level interface, and they are linked using widgets ports. Monitis²⁵ and Reveal-Cloud are two popular commercial monitoring tools that provide performance data to consumers on multiple customizable widgets.

2.8.6. Communication Protocols

Communication protocols and standards were first introduced to the industry in the late 1970's [113]. Since then, these protocols were quickly adopted for power system applications. Primarily, communication protocols were designed to ensure interoperability between multi-vendor systems [113]. These protocols also simplify integration and commissioning of data communication networks, reduce the installment costs, and allow for independent testing and validation, which in turn leads to more efficient designs.

²⁵ www.monitis.com

Communication protocols can be described as well-defined format for data exchange between communicating systems software or hardware. Each transferred data should have specific meaning to be sent or received. Besides that, a communication protocol must define a syntax, semantics and synchronization of conducted communication. Moreover, standards are adopted in order to develop communication protocols. Specialized communication protocols may be considered as extensions of simple point-to-point communication protocols [26]. Basically, layer-based architecture is frequently utilized as the basic communication protocols. Similarly, a higher layer level protocols can be designed (i.e. adding new layers of protocols) [26]. All monitoring and benchmarking commercial tools adopt communication protocols for data transfer. Communication protocols vary and are different from one monitoring tool to another. For example, Monitis and Rackspace follow HTTPs and FTP protocols. Another example is LogicMonitor, which adopts the encrypted Simple Network Management Protocol (SNMP).

2.9. Commercial and Open-Source Monitoring and Benchmarking Tools

Nowadays, there are a large number of monitoring and benchmarking frameworks and products in the market, both commercial and open-source. Commercial products usually provide comprehensive characteristics but usually they are significantly costly. Alternatively, open-source frameworks come with no cost, but they usually have some limitations, such as a limited number of devices and services to monitor or benchmark, and most significantly no technical support. This section will explore and discuss the most well-known monitoring and benchmarking products and frameworks.

2.9.1. Monitis

Monitis [115] founded in 2005, has one unified dashboard where consumers can open multiple widgets for monitoring. A Monitis consumer needs to enter his/her credentials to access the hosting cloud account. In addition, a Monitis consumer can remotely monitor any website for uptime, in-house servers for CPU load, memory, or disk I/O, by installing Monitis agents to retrieve data about the devices. A Monitis agent can also be used to collect data of networked devices in an entire network (behind a firewall). This technique is used instead of installing a Monitis agent on each single device. Widgets can also be emailed as a read only version to share the monitored information. Moreover, Monitis provides rich features for reporting the status of instances where consumers can specify the way a report should be viewed e.g. chart, or graph. It also enables its consumers to share the report publicly with others. Nevertheless, Monitis has no capabilities to apply benchmarks to the networked devices.

2.9.2. RevealCloud

CopperEgg [142, 143] provides the RevealCloud monitoring tool. It was founded in 2010 and Rackspace is a main partner. RevealCloud enables its consumers to monitor across cloud platform layers, e.g. SaaS, PaaS, and IaaS. It is not dedicated to only one cloud platform; rather it is generic to allow a consumer to get its benefits within most popular cloud providers e.g. AWS EC2, Rackspace, etc. RevealCloud is one of the very few monitoring tools that supports maintaining monitored historical data. It can track up to 30 days of historical data, which is considered as a prime feature that most commercial monitoring tools lack. However, RevealCloud lacks the functionalities to benchmark cloud resources.

2.9.3. LogicMonitor

LogicMonitor [102] was founded in 2008 and it is a partner with several third parties such as NetApp, VMWare, Dell, and HP. Similar to RevealCloud, LogicMonitor enables its consumers to monitor across cloud layers e.g. SaaS, PaaS, and IaaS. It also enables them to monitor application operations on multi-cloud resources. The protocol used in communications is SSL. Moreover, LogicMonitor uses SNMP as a method of retrieving data about distributed virtual and physical resources. The LogicMonitor framework is limited to monitoring capabilities and does not enable its users to benchmark their resources.

2.9.4. Nimsoft

Nimsoft [123] was founded in 2011. Nimsoft supports cross-layers monitoring of both virtual and physical cloud resources. Moreover, Nimsoft enables its consumers to view and monitor their resources in case they are hosted on different cloud infrastructures; for example, a Nimsoft consumer can view resources on Google Apps, Rackspace, Amazon, Salesforce.com²⁶ and others through a unified monitoring dashboard. Also, Nimsoft gives its consumers the ability to monitor both private and public clouds. However, Nimsoft does not provide any benchmarking features.

2.9.5. Nagios

Nagios [118] was founded in 2007 as an open-source network monitoring framework that is broadly used by network administrators, ISPs, governments, as well as big enterprises, (e.g. Yahoo, Amazon, and Google) [119]. Nagios is confirmed to be scalable for a large network with as many as 100,000 hosts and 1,000,000 services [119]. Technically, Nagios enables its consumers to monitor their resources on different cloud infrastruc-

²⁶ www.salesforce.com

tures as well as in-house infrastructure. Ideally, for monitoring network resources, Nagios relies on SNMP [106]. Moreover, Nagios has been extended with monitoring functionalities for both virtual instances and storage resources using a plugin-based architecture. Typically, a Nagios server is required to collect the monitoring data, which would classify it as a centralized solution. However, many possible configurations can help create multiple hierarchical Nagios servers to reduce the disadvantages of a centralized server. A large number of plug-ins are available from the Nagios library, which means a Nagios user can customize functionalities accordingly with requirements [23].

2.9.6. SPAE by SHALB

SHALB [162] was founded in 2002 and provides a monitoring solution called Security Performance Availability Engine (SPAE). SPAE is a typical network monitoring tool supporting a variety of network protocols such as HTTP, HTTPS, FTP, SSH, etc. It uses SNMP [161] to perform all of its monitoring processes and emphasizes security monitoring and vulnerability. However, SPAE does not support cross-layer monitoring (IaaS, PaaS and SaaS).

2.9.7. Amazon CloudWatch

CloudWatch [67] is one of the most popular commercial tools for monitoring cloud resources. It is provided by Amazon and it resides on EC2 to enable its consumers to monitor their resources and to measure the level of usage [67]. Namely, Cloudwatch can monitor Amazon resources such as Amazon EC2 instances, Amazon DynamoDB tables, and Amazon RDS DB instances, as well as custom metrics generated by the user applications and services, and any log files these applications may generate [67]. However, Cloudwatch does not support multi-cloud infrastructure monitoring but it has limited

infrastructure level metrics. Further, the Cloudwatch can keep up to only two weeks of data history [129]. The user interface of Cloudwatch limits the user to view only metric at a time. Moreover, the technical approaches used in CloudWatch to collect data are implicit and not exposed to users. However, an API is provided for users to collect metrics at any cloud layer but requires the users to write additional code. Furthermore, Cloudwatch does not provide benchmarking capabilities for its users.

2.9.8. OpenNebula

OpenNebula [112] is an open source monitoring system that provides the ability to manage the complexity and heterogeneity of large and distributed infrastructures [126]. It uses SSH as the protocol permitting consumers to gain access and gather information about their resources [126]. Mainly, OpenNebula is concerned with monitoring physical infrastructures involved in data centers such as private clouds. OpenNebula toolkit manages a data center's virtual infrastructure to build private, public and hybrid implementations of infrastructure as a service [114]. Hence, OpenNebula is one of the most commonly used open-source cloud management frameworks among research institutions and enterprises.

2.9.9. CloudHarmony

CloudHarmony [156] started monitoring services in the beginning of 2010. It provides a set of performance benchmarks of public clouds. It is mostly concerned in monitoring the common operating system metrics that are related to (CPU, disk and memory). Moreover, cloud to cloud network performance in CloudHarmony is evaluated in terms of RTT and throughput.

2.9.10. Windows Azure FC

Azure Fabric Controller (Azure FC) [57, 58] provided by Microsoft adopts a centralized network architecture. It applies cross-layers monitoring on Azure platform but, does not support monitoring on multi-clouds. Moreover, Azure FC utilizes SNMP to perform monitoring. Similar to Cloudwatch, the technical approaches used in Azure FC to collect data are implicit and not exposed to users.

2.9.11. Lattice Framework

In [37], the Lattice monitoring framework is presented for monitoring virtual and physical resources. The Lattice framework is able to collect the information for CPU usage, memory usage, and network usage of each Virtual Execution Environment (VEE) and VEE host which is the physical resource. Nevertheless, Lattice lacks the functionality of benchmarking cloud resources.

2.9.12. QoS-MONaaS

MONitoring as a Service (QoS-MONaaS) [145] [3] is a monitoring framework. It consists of a cloud-based application which runs on top of a Stream Routing Technology for 2015 (SRT-15) platform, hence, QoS-MONaaS is exposed as a web service [145]. 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.).

2.9.13. PCMONS

A monitoring framework known as (PCMONS) is developed by incorporating previous frameworks and techniques [86]. Mainly, PCMONS is developed for private clouds environments and is focused on virtual machines (PaaS layer). PCMONS proves that cloud

computing is a viable way of optimizing existing computing resources in datacenters. Also, with PCMONS, orchestrating monitoring solutions on installed infrastructures becomes viable.

2.9.14. SBLOMARS

A number of frameworks have been proposed for VM management (PaaS layer), which includes Simple Network Management Protocol (SNMP) for data retrieval. SBLOMARS [105] 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 its monitoring tasks, SBLOMARS focuses on enabling multi-constrain resource scheduling in grid computing environments.

2.9.15. Apache CloudStack

CloudStack [80] is an open source software platform that focuses on IaaS of the cloud layers. It provides monitoring capabilities to manage the large network of virtual machines, storage, and compute nodes that make up a cloud infrastructure. Furthermore, CloudStack can run multiple hypervisors of multiple types, support multi-tenancy and account management, and provide an easy-to-use web interface [80]. CloudStack is written by Java and it has a command line mode and API mode based on REST [28]. Nevertheless, CloudStack does not enable monitoring applications across-layers nor benchmarking capabilities.

2.9.16. Compuware's Gomez

Compuware's Gomez [176], is a solution for web performance optimization, (revenue based web and mobile applications). Gomez focuses on monitoring web applications

(SaaS layer) from the end users' perspective. Moreover, it focuses on cross-browser testing and web load testing to optimize web-site performance. However, Compuware lacks in its ability to monitor fine-grained application server and database events across multiple layers of the cloud [32].

2.9.17. Cloud Object Storage Benchmark (COSBench)

COSBench [187] is a tool that is designed and implemented in Intel for benchmarking cloud object storage services. However, COSBench lacks the monitoring capabilities required for applications components. Moreover, COSBench is restricted to cloud storage services and therefore can only measure three metrics.

2.9.18. C-MART

For applications benchmarking C-MART [170] has a notable effect on the design and the simplicity of a web application benchmarking tool. C-MART presents a significant tool to emulate and then benchmark web applications such as online store or social networking websites. Originally, C-MART is motivated by the fact that benchmarks need to cope with the shift from the traditional environments to cloud environments. However, C-MART is limited to benchmarking applications at the PaaS layer and resources at IaaS layer.

2.9.19. CloudGauge

CloudGauge [168], presents an effective dynamic virtual machine (PaaS layer) benchmarking tool. It provides automated scripts to provision and measure the performance of the virtual environment setup. But, the focus of CloudGauge experimental benchmark was on the virtualization layer. Furthermore, the data collected was mainly CPU usage and average load memory.

2.9.20. CLIQr

CliQr [38] is a benchmarking framework for optimizing cloud applications performance. CliQr focuses on parameters pertaining to cost-performance among cloud platforms. Hence, CliQr supports several cloud platforms such as Amazon, Microsoft Azure, and Rackspace. However, CliQr does not consider other QoS parameters and lacks monitoring capabilities. Moreover, CLIQr does not support cross-layers benchmarking.

2.9.21. Hawk-I

Hawk-I [117] is a project to study different VMs behaviors on cloud platforms. Mainly, the implementation of Hawk-I was done on Amazon EC2. Hence, Hawk-I focuses only on VMs (PaaS layer) but not on cloud applications (SaaS layer). Moreover, Hawk-I does not enable users to apply monitoring functions.

2.9.22. mOSAIC Benchmark

mOSAIC Benchmark [17] enables users to compare different cloud providers offerings to host applications. It stresses custom benchmarks for its users to be applied on different cloud platforms. Similar to Hawk-I, mOSAIC Benchmark focuses on VM instances (PaaS layer) for benchmarking processes. However, it does not provide cross-layers benchmarking. Further, it does not enable monitoring functionalities.

2.10. Classification and Analysis of Cloud Monitoring and benchmarking tools - Gap Analysis

With increasing cloud complexity, efforts needed for monitoring and benchmarking cloud resources need to be multiplied. The size and scalability of clouds when compared

to traditional infrastructures involve more complex monitoring and benchmarking frameworks that have to be scalable, effective and fast. Technically, this would mean that there is a demand for real-time reporting of performance measurements while monitoring and benchmarking cloud resources and applications. Therefore, cloud monitoring and benchmarking frameworks need to be advanced and customized according to the diversity, scalability, and high dynamic cloud environments.

In section 2.8, the main evaluation dimensions of monitoring and benchmarking frameworks are presented. As discussed, not all of those dimensions are adopted by monitoring and benchmarking frameworks in either open source or commercial domains. However, most of these dimensions, which are basically related to performance, have been addressed by the research community and have received some attention; more considerable effort to achieve higher level of maturity is essential for cloud monitoring and benchmarking frameworks.

Decentralized approaches are gaining more trust over centralized approaches. In contrast to unstructured P2P, structured P2P networks present a practical and more efficient approach in terms of network architecture. Considering interoperability, either cloud-dependent or cloud-agnostic, both of these monitoring and benchmarking approaches have gained high importance. Currently, both approaches are supported by several monitoring and benchmarking frameworks. This research finds that cloud-dependent monitoring and benchmarking frameworks are mostly commercial; whereas cloud-agnostic monitoring and benchmarking frameworks are typically open source.

In addition, the matrix of the quality of service is the most important dimension of monitoring and benchmarking frameworks. I elaborate on how those quality parameters should be monitored, detected and reported and at which cloud layer a monitoring and benchmarking frameworks should operate their processes. Further, the aggregation of multiple parameters for a consumer application is a critical aspect of monitoring and benchmarking. This means that a monitoring and benchmarking framework should not be cloud layer specific or layer agnostic. This will determine the visibility characteristic of a cloud monitoring and benchmarking framework. All of these issues in monitoring and benchmarking need more study by the cloud community and are still in demand for more technical improvements. Table 2.2 summarizes this research of monitoring and benchmarking frameworks against evaluation dimensions explored in Section 2.8. Moreover, Figure 2.13 presents a summary of all evaluation dimensions that were discussed in section 2.8.

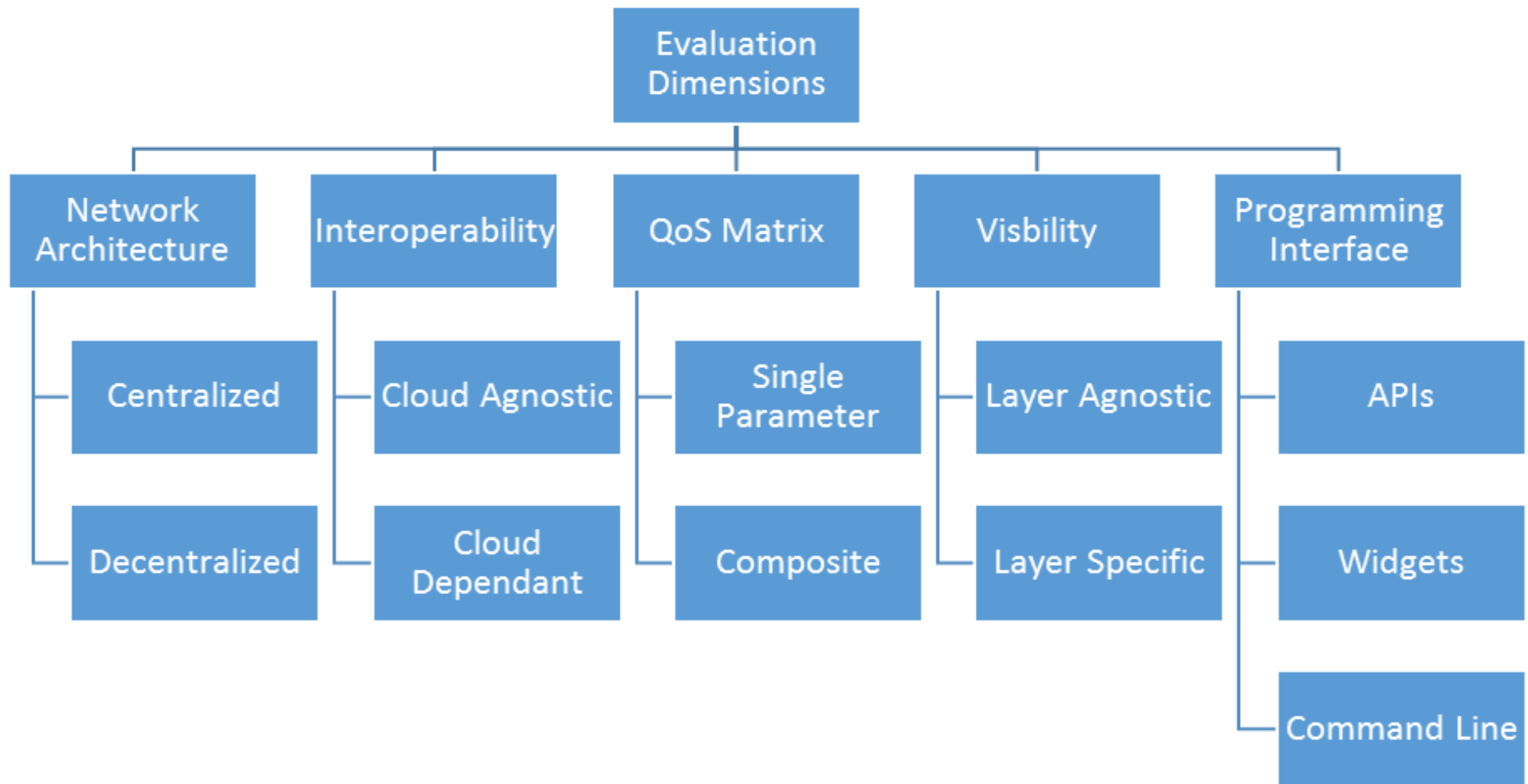


Figure 2.13: Evaluation Dimensions tree diagram.

Framework	Network Arch. (Centralized)	Network Arch. (Decentralized)	Interoperability Multi-Cloud	Visibility Cross-Layers	SNMP	Extendable APIs	Benchmarking
Monitis [115]	Not-Stated (SaaS solution)	Not-Stated (SaaS solution)	✓	✓	✓	✓	X
RevealCloud [142, 143]	Not-Stated (SaaS solution)	Not-Stated (SaaS solution)	✓	✓	Not-Stated	✓	X
LogicMonitor [102]	Not-Stated (SaaS solution)	Not-Stated (SaaS solution)	✓	✓	✓	✓	X
Nimsoft [123]	✓	Not-Stated	✓	✓	✓	✓	X
Nagios [118]	✓	Not-Stated	✓	✓	✓	✓	X
SPAE [162]	Not-Stated (SaaS solution)	Not-Stated (SaaS solution)	✓	X	✓	X	X
CloudWatch [40]	Not-Stated (SaaS solution)	Not-Stated (SaaS solution)	X	✓	Not-Stated	✓	X
OpenNebula [112]	✓	X	X	X	Not-Stated	X	X
CloudHarmony [156]	Not-Stated (SaaS solution)	Not-Stated (SaaS solution)	✓	X	Not-Stated	X	✓
Azure FC [57, 58]	✓	Not-Stated	X	✓	✓	✓	X
SBLMARS [105]	✓	X	✓	X	✓	X	X

COSBench [187]	✓	X	✓	X	X	✓	✓
CliQr [38]	Not-Statesd (SaaS solution)	Not-Statesd (SaaS solution)	✓	X	X	✓	✓
Hawk-I [117]	Not-Statesd (SaaS solution)	Not-Statesd (SaaS solution)	✓	X	X	X	✓
mOSAIC Benchmark [135]	Not-Statesd (SaaS solution)	Not-Statesd (SaaS solution)	✓	X	X	✓	✓
Compuware's Gomez [176]	Not-Statesd (SaaS solution)	Not-Statesd (SaaS solution)	✓	X	X	X	✓
C-MART [170]	Not-Statesd (SaaS solution)	Not-Statesd (SaaS solution)	✓	X	X	✓	✓
CloudGauge [55]	Not-Statesd (SaaS solution)	Not-Statesd (SaaS solution)	✓	X	X	X	✓
CLAMBS	✓	✓	✓	✓	✓	✓	✓

Table 2-2: Summary of studied monitoring and benchmarking frameworks.

2.11. Summary

To investigate existing research in cloud applications monitoring, this chapter has reviewed the background literature on applications management in cloud environments. In addition, it discussed the state-of-the-art research in the area of cloud monitoring. Also, it presented the evaluation dimensions which, by monitoring tools, can be studied and evaluated. Furthermore, cloud benchmarking was given specific attention in this review. In doing so, this chapter highlighted several design issues and research dimensions that can help to evaluate cloud application monitoring and benchmarking frameworks. Moreover, this chapter explored and presented several cloud monitoring and benchmarking tools, their features and shortcomings. Finally, the chapter presented an evaluation framework of current cloud monitoring tools.

3. Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework

3.1. Introduction

A major focus of this thesis is to address the key challenges in performance monitoring and benchmarking of cloud-based applications deployed across multiple cloud providers and platforms.

This chapter proposes *CLAMS: Cross-Layer Multi-Cloud Application Monitoring as a Service Framework*. The CLAMS framework enables the monitoring of applications components (e.g. streaming server, web server, indexing server, compute service, storage service, and network) across-layers (Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS)) on multi-clouds. CLAMS allows effective capturing and sharing of QoS information. It employs an agent-based approach that is cloud provider environment agnostic, i.e. making it compatible with any cloud provider.

The CLAMS framework stands as a solution framework to address the gaps in existing monitoring approaches and a mechanism for improved cloud applications monitoring. More specifically, CLAMS aims to address the following gaps in current monitoring approaches:

- Performance monitoring for cloud applications' components that are distributed/hosted across different layers of the cloud platforms (e.g. PaaS, IaaS). Research in this area of cloud applications monitoring is still in its infancy. Current approaches do not allow cross-layers monitoring, hence lacking the visibility required to monitor QoS of individual components of the application (e.g., web server component, database server component). The most monitoring approaches are layer-specific.
- Current approaches lack the ability to monitor and profile QoS of applications, whose parts or components are distributed across multiple heterogeneous public or private clouds, namely, multi-clouds. They do not enable the users to monitor their distributed application's components on multi-clouds.
- Current approaches lack effective collection and sharing of QoS information across cloud layers using a cloud provider agnostic technique;

Summing up, current monitoring approaches do not support monitoring across cloud platforms as well as cloud platform layers. Hence, there is a need for a monitoring framework that has the capability to perform cross-layer monitoring as well as work across multiple cloud providers in a coordinated manner to deliver QoS requirements of distributed cloud applications.

The chapter is organized as follows. Section 3.2 presents an overview of CLAMS framework and the CLAMS data collection model. Section 3.3 presents the CLAMS architecture components. Section 3.4 illustrates the visibility and interoperability in

cross-cloud monitoring on multi-clouds. Section 3.5 discusses the use of CLAMS with a real-world application scenario. Section 3.6 presents a comparison between the CLAMS framework and current monitoring frameworks. Finally, section 3.7 concludes this chapter. Parts of this chapter have appeared in publications (K Alhamazani et al., 2014b), (K Alhamazani et al., 2014c).

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

As argued in previous sections, monitoring is vital for: i) managing the QoS of 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.

3.2.1. General Overview

Figure 3.1 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 for monitoring resource/application's component information at various layers including SaaS, PaaS and IaaS. A Manager Agent is responsible to collect QoS data from each Monitoring Agent.

CLAMS includes effective mechanisms for efficient cloud monitoring at different *aaS layers. It provides standard interfaces and communication protocols that enable an application/system administrator to gain awareness of the whole application stack across different cloud layers in heterogeneous environments (monitor VMs hosted on different cloud platforms). In this way, CLAMS also satisfies the challenges related to interoperability between heterogeneous cloud resources.

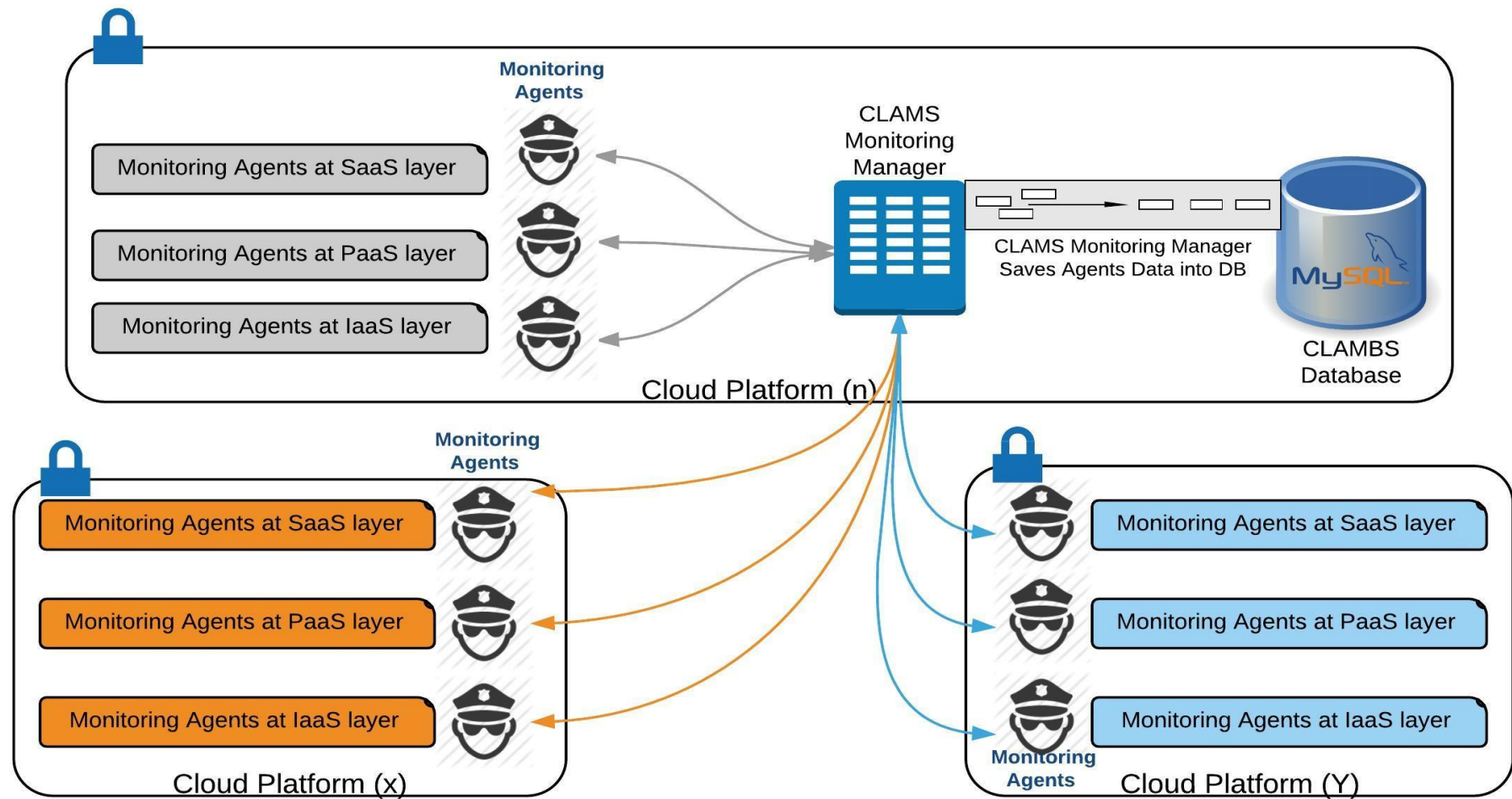


Figure 3.1: CLAMS Framework overview.

The CLAMS Monitoring Agents reside on various cloud layers e.g. IaaS, PaaS, and SaaS. At each layer, one or more CLAMS Monitoring Agents can be allocated according to the monitored components. For example, in figure 3.2, one CLAMS Monitoring Agent can be allocated to monitor the web server component at PaaS layer. Similarly, another CLAMS Monitoring Agent at IaaS layer can be allocated to monitor the storage component. The CLAMS Monitoring Manager and the CLAMS database component can reside on the same cloud platform where CLAMS Monitoring Agents are running or they can be hosted on a different cloud platform. In addition, a CLAMS Super-Manager is incorporated to support the interoperability feature of the CLAMS framework.

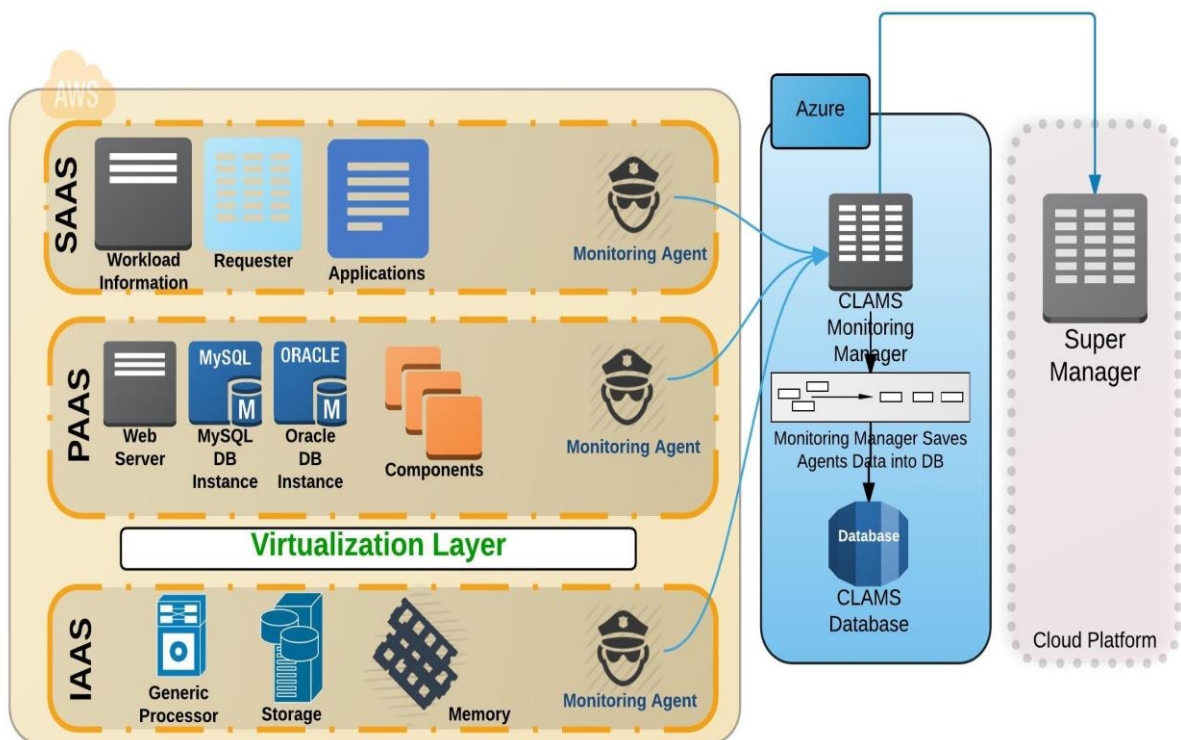


Figure 3.2: CLAMS distributed components.

3.2.2. CLAMS Data Collection Model

The CLAMS framework also includes a data collection model. This model classifies the different collected QoS performance parameters. The QoS parameters are presented and classified in Table 3.1 for cloud applications (e.g., multi-tier web applications, content delivery networks, etc.) in general. Also, this table shows an application's components at each layer (Web Server, Streaming Server, Indexing Server at PaaS layer, Compute resource, Storage resource at IaaS layer) and QoS parameters for these components at each layer (SystemUpTime, SystemServices at PaaS layer, SpeedClock, RoundTripDelay at IaaS layer). Furthermore, figure 3.3 shows the Entity Relationship (ER) diagram for the data presented in Table 3.1.

Cloud Layer	Layer Applications' Components	Targeted QoS Parameters	
SaaS	User Applications (Servers App. Web App, Microsoft Word. etc)	No. BytesRead No. BytesWrite Availability CPU Utilization Mem Utilization	
PaaS	Web Server, Streaming Server, Indexing Server, Apps Server, etc	SystemUpTime SystemServices SystemDesc Utilization	
IaaS	Compute resource, Storage resource, Network, etc	CPU	Network
		SpeedClock CurrentState Utilization	Bandwidth OnewayDelay RoundTripDelay TcpConnState

Table 3-1: QoS parameters for relative cloud platform layers.

Furthermore, figure 3.3 shows the Entity Relationship (ER) diagram for the data presented in Table 3.1. To illustrate, one block of the diagram shown in figure 3.3 represents the cloud platforms layers (e.g. SaaS, PaaS and IaaS). The second block of the diagram represents the application's components that relate to each layer. That is, for the SaaS layer we may have various applications' components (e.g. Servers App, Web App, Microsoft Word, etc). Likewise, at the PaaS layer will have other components (e.g. Web Server, Streaming Server, Indexing Server, Apps Server, etc). Similarly, the IaaS layer will have different components (e.g. Compute resource, Storage resource, Network, etc). The last block of the diagram links various QoS parameters to different components at each layer. That is, for the aforementioned applications' components at SaaS layer we may have different QoS parameters (e.g. No. BytesRead, No. BytesWrite, Availability, CPU Utilization, Mem Utilization). Likewise, for the aforementioned applications' components at PaaS layer we may have different QoS parameters (SystemUpTime, SystemServices, SystemDesc, Utilization). Similarly, for the aforementioned applications' components at IaaS layer we may have different QoS parameters (e.g. SpeedClock, CurrentState, Utilization, One-wayDelay, RoundTripDelay).

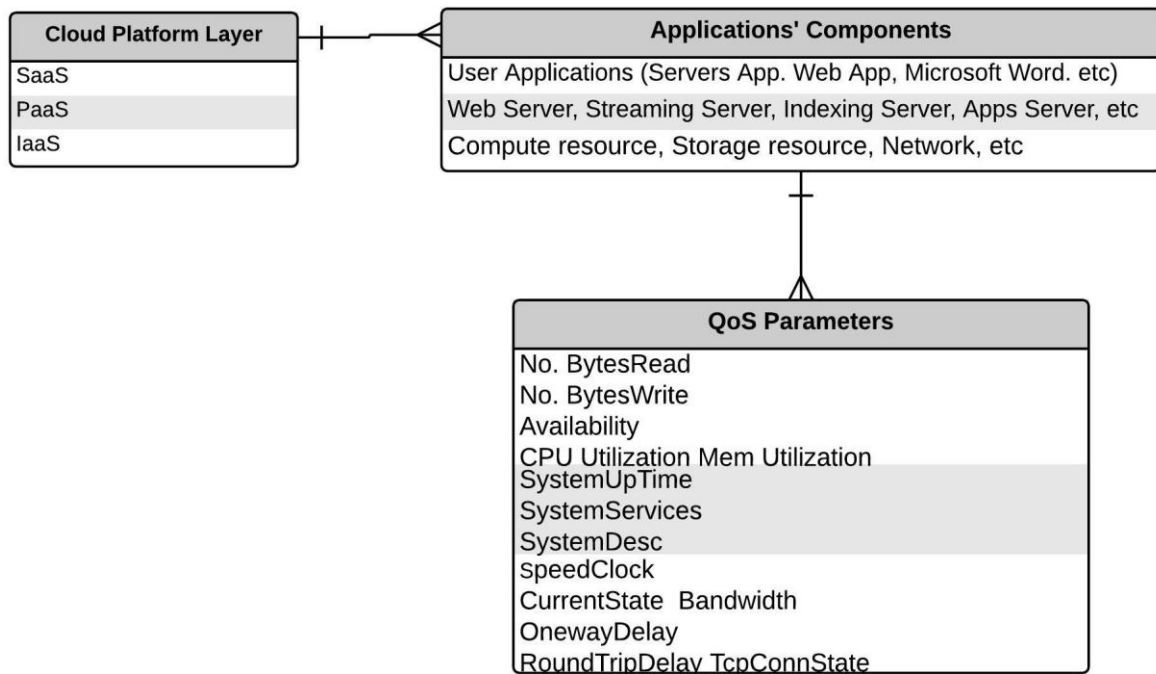


Figure 3.3: ER for the cloud layer, applications' components, and QoS parameters.

3.3. CLAMS Architecture Components

As illustrated in figure 3.2 (section 3.2), the CLAMS framework comprises three main components namely, CLAMS Monitoring Super-Manager, CLAMS Monitoring Manager and CLAMS Monitoring Agent.

3.3.1. 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 commu-

nication between the Monitoring Manager and the Monitoring Agent can employ a push or pull technique. In case of a pull technique, the Monitoring Manager pulls the CLAMS Monitoring Agents at different frequencies, and 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 [137], 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 are collected by the Manager from the Agents. Figure 3.4, presents the CLAMS Monitoring Manager component and the Agent component of the CLAMS framework.

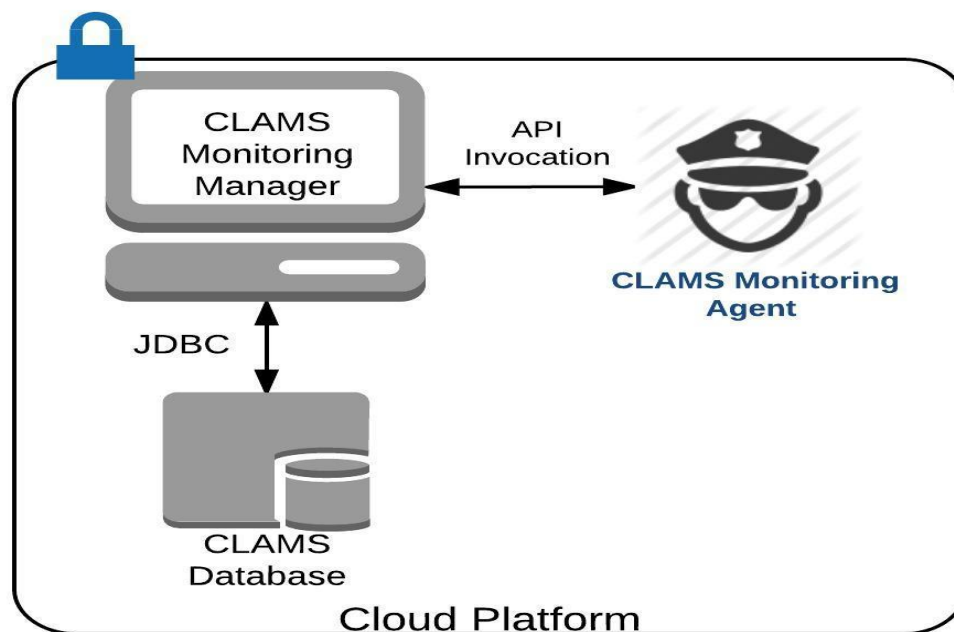


Figure 3.4: CLAMS Monitoring Manager component and CLAMS Monitoring Agent components.

To illustrate further, consider an audio/video streaming application running on the cloud which has several distributed components (e.g. web server and an indexing server) at the PaaS layer and (e.g. storage server) at IaaS layer. Each component of such application is running and hosted on different VMs. The web server has an IP address say, 192.168.1.1, the 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 application's components and VM parameters. In this case, the Monitoring Manager can send a first request to the Monitoring Agent on the web 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 Monitoring 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 Monitoring 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). Figure 3.5, presents a pseudo code describing the interaction process between the CLAMS Monitoring Manager and the distributed CLAMS Monitoring Agents.

Method Connect_To_Monitoring_Agents() :

```

Connect to Monitoring_Agent on <Web server>
    Request for <CPU utilization>
    Get <Responses>
Connect to Monitoring_Agent on <Indexing server>
    Request for <Packets In>
    Get <Responses>
Connect to Monitoring_Agent on <Storage server>
    Request for <actual used memory>
    Get <Responses>
Save <Responses> into DB

```

Figure 3.5: Interaction of the CLAMS Monitoring Manager and distributed Monitoring Agents – Pseudo Code.

The CLAMS Monitoring Manager employs the QoS data collection schema (described in section 3.2.2) to store QoS statistics into the local database and an agent schema to maintain the list of discovered agents. The CLAMS Monitoring Manager also incorporates an API that is used by another Monitoring Manager or external service to share the QoS statistics.

3.3.2. CLAMS Monitoring Agent

Another major component of the CLAMS framework is the Monitoring Agent. The Monitoring Agent resides on a VM running on the cloud collects and sends QoS parameters' values as requested by the CLAMS Monitoring Manager. After the monitoring system initialization, the Monitoring Agent waits for the incoming requests from the manager or starts to push QoS data to the Monitoring Manager. Upon arrival of the request, the Monitoring 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 Monitoring Manager.

In addition, 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. Moreover, the proposed blueprint does not restrict future developers from extending CLAMS to their purposes. In the proof-of-concept implementation explained in chapter 5, a combination of SNMP and RESTful web services have been used. 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. Figure 3.6, presents a pseudo code describing the startup and interaction of the Monitoring Agent.

```

Start Monitoring_Agent
Read <configuration>

While(monitring process = on)
    Collect data and store locally
    If communication Type = <proactive>
        Monitoring_Agent send data to Monitoring_Manager
    Else communication Type = <reactive>
        Monitoring_Agent wait for request from Monitoring_Manager
        Send collected data to Monitoring_Manager
    End_If
End_while

```

Figure 3.6: The CLAMS Monitoring Agent startup and monitoring process – Pseudo Code.

3.3.3. CLAMS Super-Manager

The CLAMS Super-Manager is a software component that collects QoS information from CLAMS distributed Monitoring Managers running across multi-cloud environments. To achieve heterogeneity and multi-cloud functionality, a hierarchical approach can be applied using the Super-Managers as depicted in figure 3.7. The function of a Super-Manager is marginally different from a Monitoring Manager.

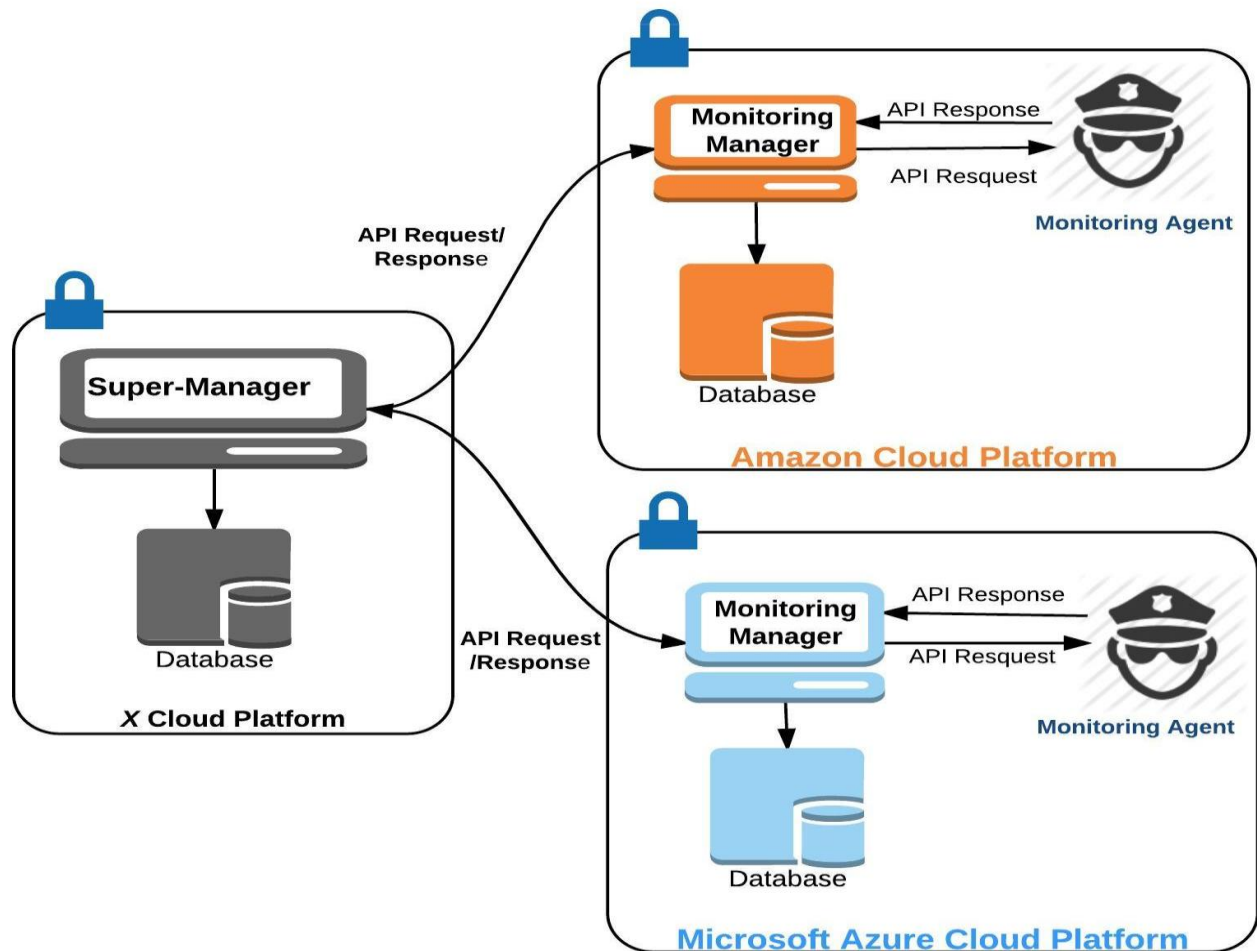


Figure 3.7: CLAMS hierarchical approach.

The Super-Managers are responsible for coordinating between multiple Monitoring Managers using the monitoring manager's API. The Monitoring Managers (depicted as managers) will retrieve the monitored data from Monitoring Agents, and then they will re-send the data to the Super-Managers. In a wider scope, a hierarchy of Super-Managers can be formed where a Super-Manager instance can collect data from multiple Super-Managers instances, see figure 3.8.

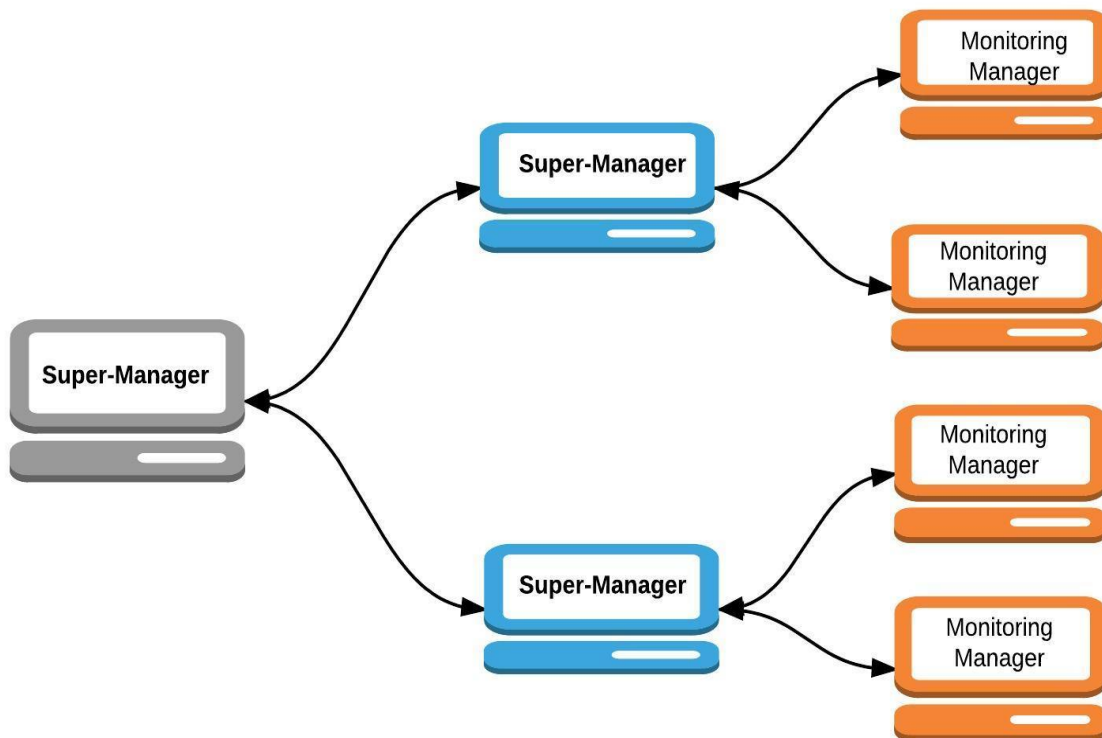


Figure 3.8: CLAMS Super-Manager hierarchy approach – wider scope.

Figure 3.9 presents the procedure of interaction where the CLAMS framework incorporates a Super-Manager component. In the figure, the CLAMS components are distributed on multi-cloud platforms, namely, Amazon EC2, and Microsoft Azure platforms. On the Azure platform, the CLAMS Monitoring Manager sends a request to the CLAMS Monitoring Agent for QoS parameter (e.g. CPU utilization). The CLAMS Monitoring Agent responds back with a value of the CPU utilization to the CLAMS Monitoring Manager. After that, the CLAMS Monitoring Manager sends the retrieved response to the CLAMS Super-Manager, which resides on a different cloud platform, namely, Amazon platform.

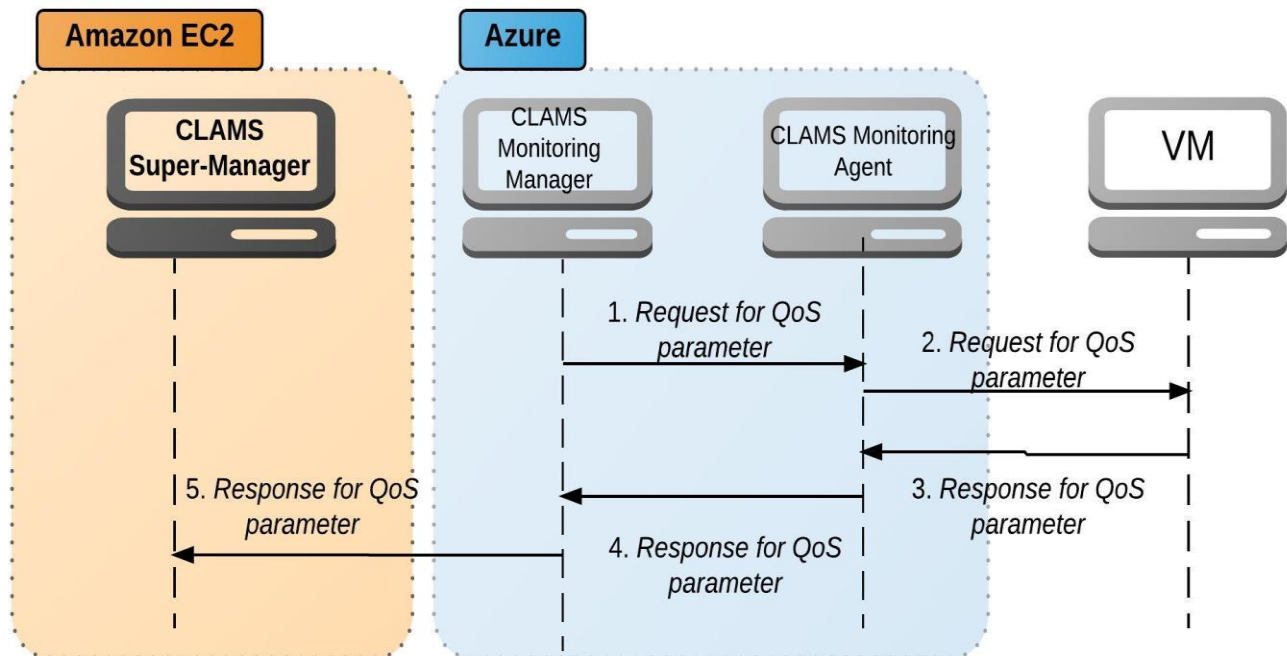


Figure 3.9: Interoperable CLAMS components communication.

3.4. Visibility and Interoperability

In this section, we will first discuss the monitoring across cloud platforms layers. This monitoring process makes all application's components visible during monitoring. Second, I will discuss the interoperable feature of CLAMS where monitoring is applied to distributed application's components across multi-cloud platforms.

3.4.1. CLAMS: Cross-layers monitoring (Visibility vs. Black Box View)

Further, the CLAMS framework aimed is to be cloud layer agnostic. That is, CLAMS distributed Monitoring Agents can operate across cloud platform layers individually. The QoS monitoring problem is cross-cutting as it extends across the multiple

cloud platform layers. For example, failure of VM (IaaS offering) affects the QoS of a web application container (PaaS offering) or a database application (PaaS offering) container hosted within that VM. This ultimately affects the end user QoS of end-user of that web application (SaaS offering). A cloud provider can manage and administer the cloud-resources/applications more efficiently and effectively when it gains in-depth status information of the application components cross-layers individually as against a black box view.

For example, an application such as WordPress²⁷ will typically have a MySQL database and an Apache Web Server as the underlying components. Current cloud application monitoring frameworks like Amazon CloudWatch²⁸ 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 cross-layers. Figure 3.10 demonstrates how CLAMS Monitoring Agents are distributed to monitor the Wordpress application's components, namely, Apache Tomcat, and MySQL. The figure shows a dedicated CLAMS Monitoring Agent for each of the Wordpress application's components. Also, CLAMS Monitoring Agents are distributed to monitor the IaaS resources (e.g., CPU resource).

²⁷ <https://wordpress.org/>

²⁸ <http://aws.amazon.com/cloudwatch/>

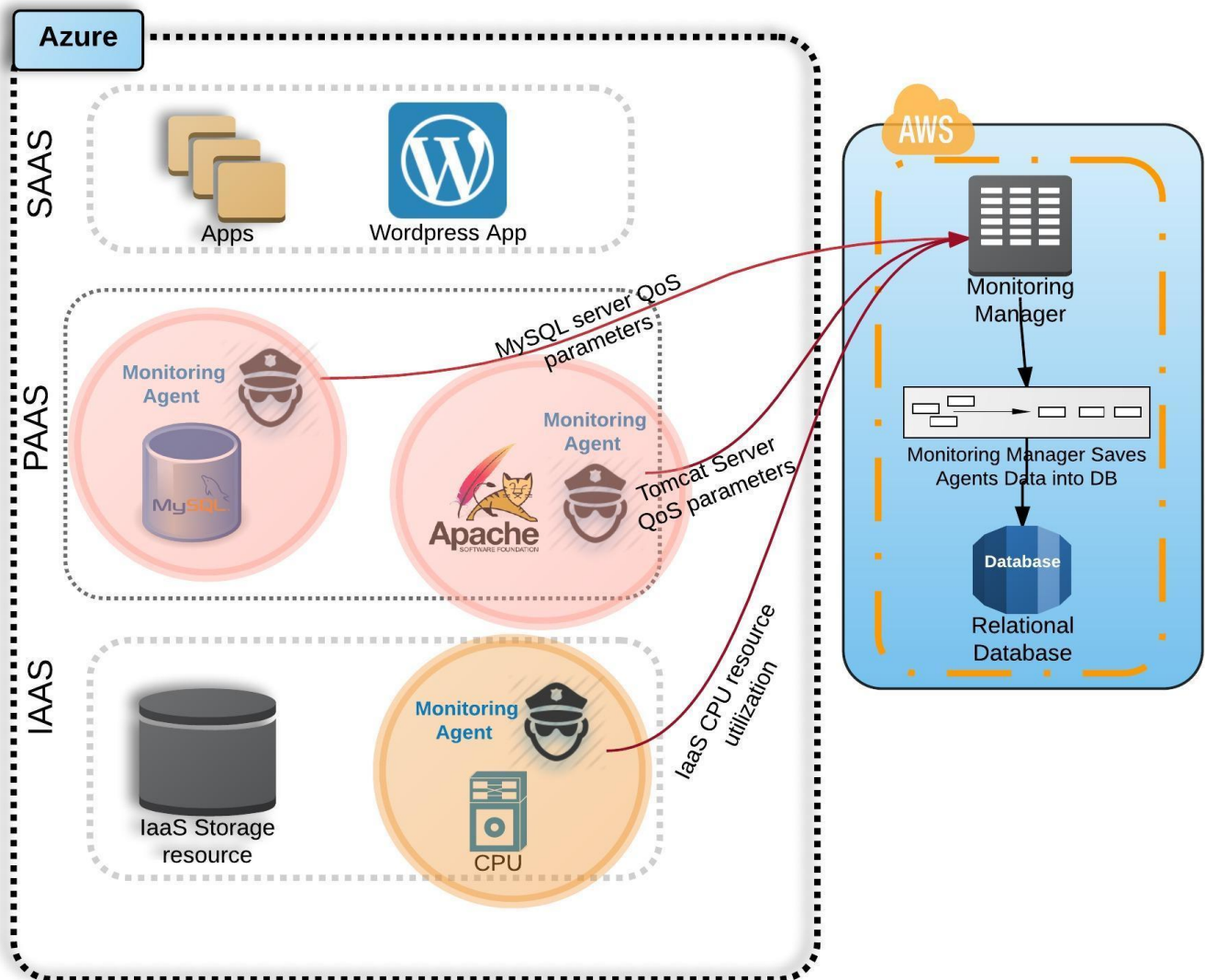


Figure 3.10: Cross-layers monitoring (Visibility).

3.4.2. CLAMS: Hierarchical Support for Multi-Cloud Environments (Interoperability)

Further, the CLAMS monitoring framework is cloud layer and provider agnostic. That is, the CLAMS Monitoring Manager/Agent may run on heterogeneous cloud platforms. Current cloud monitoring frameworks are mostly incompatible/ineffective in multiple cloud provider environments. For example, Amazon

CloudWatch does not allow monitoring application components hosted on non-Amazon platforms. This defeats the distributed nature of cloud application hosting.

In case the monitoring 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 both of these cloud platforms. Figure 3.11 illustrates the interoperable CLAMS framework advantageous ability through its components. In the figure, the Wordpress application's components are distributed on multi-cloud platforms, namely, Amazon and Azure platforms. The CLAMS Monitoring Agents as shown in the figure are distributed and dedicated to individual components (e.g., Apache Tomcat, and MySQL). This enables the CLAMS Monitoring Manager to collect QoS performance information from these distributed application's components while they are hosted on two different cloud platforms. Moreover, other CLAMS Monitoring Agents are allocated to monitor resources components at IaaS layer (CPU performance).

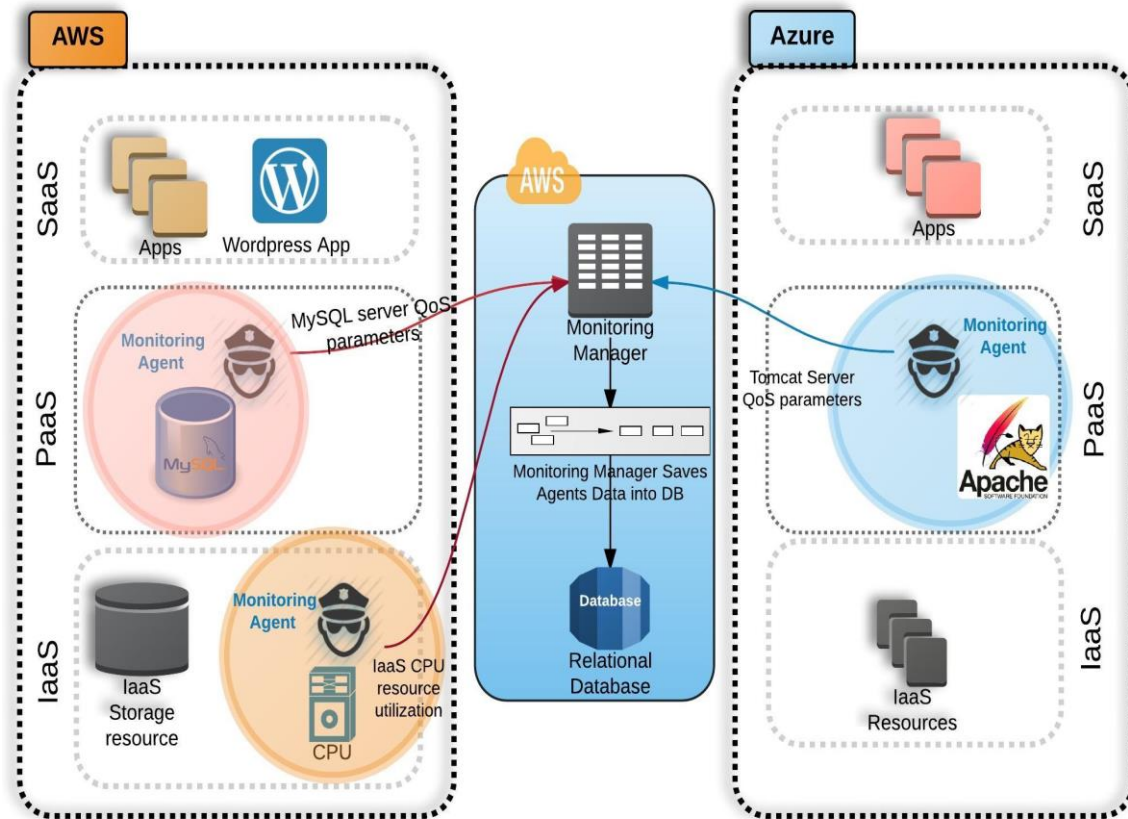


Figure 3.11: Cross multi-clouds monitoring (Interoperability).

The CLAMS framework components, namely, the CLAMS Super-Manager, Monitoring Manager, and the CLAMS Monitoring Agent exemplify the advantageous features of CLAMS framework, which provides a uniform, extensive and effective cross-layer monitoring. The CLAMS framework aids in controlling the application QoS based on real-time monitoring of the status of application components and underlying cloud platform (hardware and software).

3.5. CLAMS Applications Scenario

Emerging trends in big data analytics supported by advances in cloud computing have shifted the focus from “What data should we store” to “What can we do with the data” [36] leading to Analytics-as-a-service model. Big data analytics offer valuable insight into data that can offer a competitive advantage to organizations.

To support an Analytics-as-a-service model in the cloud specifically in environments where floods of data generated from smart phone and sensors are increasing by the day and unpredictable. Big data analytics in the cloud require real-time QoS monitoring across the cloud layers to ensure an application’s availability and performance. Consider an example of a crowd-sensing application that is supported by a distributed big data analytics application (Hadoop + Mahout) in the cloud. The volume and variety of data depends on the number of users contributing data which in most cases is not known before-hand. Hence, it is essential to continuously monitor individual system performance at different cloud platform layers such as Hadoop job tracker (PaaS offering), HDFS layer (PaaS offering) and VM performance/failure (IaaS offering). The above QoS parameters cumulatively affect the QoS of the end-user of the big data analytics application (SaaS offering).

3.5.1. Big Data Analytics Application Scenario

To illustrate the need and function of the proposed CLAMS framework, consider a scenario of the Emergency Situation Awareness (ESA) as depicted in figure 1.2 in chapter 1. Systems such as ESA are required to efficiently manage and respond to situations like public demonstrations, interior domestic clashes, revolutions, major festivals, and major public/national events. The system is a typical example of a big

data analytics system including functions such as continuous data mining on data obtained from crowds and social media to detect potential danger, and machine learning algorithms to predict future occurrences of events i.e. modeling of event outcomes based on current data etc. In such situations, sidelining of danger and emergency cases cannot be predicted. Hence, an immediate and continuous monitoring is required by authorities e.g. Police, and National Guards.

To support a system such as ESA (chapter 1, figure 1.2) that requires a round-the-clock, figure operation, robust techniques are needed to ensure system performance and availability. Based on historical inferences, ESA systems use certain policies and procedures to define SLAs. Such policies and procedures are formulated to handle/avoid ESAS bottlenecks in terms of known QoS parameters e.g. network traffic hazards, VM failures etc. Furthermore, consider the following situations which foster the need for cross-layers monitoring on multi-clouds for optimized provisioning of big data analytics applications.

In some events, public users contributing data may increase phenomenally, providing valuable inputs related to that event (Step 1), see (chapter 1, figure 1.2). Such a burst of input data is hard to estimate or predict. Moreover, dynamic changes in situation might require additional machine learning algorithms to be deployed on-demand to process and filter incoming data. The ESA system will be required to cope with such dynamic demands to changing data patterns maintaining high level of system stability and availability.

3.5.2. How we detect failures using a Conventional Approach

When events described previously occur (section 3.5.1), the demand on the system increases significantly. Furthermore, in such situations, the interactions between the system and other users e.g. police, hospital etc. also increase as more critical events are identified by the big data analytics algorithm running in the cloud. To cope with such dynamic situations, monitoring the entire VM as a black-box is not sufficient to guarantee a systems' SLA. Moreover, current-monitoring approaches do not provide an insight detection of failures sources. Rather, current approaches will only provide a general non-holistic view of the application performance. The application administrator is not able to inspect each failure origin precisely.

As illustrated in figure 3.12, for an application's distributed components, there are different QoS parameters at each relative cloud platform layer, for example, *CPU* and *network* QoS parameters at the IaaS layer; *SystemProcesses*, *SystemUpTime*, *SystemDesc* at PaaS layer, *Availability*, *Delay* at SaaS layer.

Knowing individual component performance accurately greatly helps in auto-scaling the corresponding layer at the right timing to avoid failure. To illustrate, when input data load increases, the load on the corresponding system components increase (e.g. queuing, Mahout, HDFS), which may lead to system failure. If the monitoring approach being adopted cannot do cross-layers monitoring, none of the aforementioned QoS parameters will be detected specifically as a failure source or failure causing component. Monitoring the whole application does not lead to the detection of the exact cause of any existing failure at which cloud layer. For example, in case of application distributed components, the CPU performance statistics rela-

tive to the application will not determine on which platform it has occurred. Thus, the required scaling and provisioning may not detect the issue until it impacts the entire VM.

Therefore, current monitoring approaches encounter two critical monitoring issues:

- 1) Current monitoring approaches cannot detect at which cloud layer the failure occurs. That is, the exact distributed application component which causes the failure is not detected.
- 2) Current monitoring approaches cannot detect on which cloud platform the failure exists. This means, the monitoring tool is not capable to determine on which cloud platform the arising failure exists.

3.5.3. Using CLAMS to detect and identify at which cloud layer a failure occurs (Visibility)

CLAMS, in such failure (monitoring issue 1, section 3.5.2) events, helps in identifying and rectifying a specific QoS parameter which leads to the exact component. In figure 3.12, components are associated with cloud layers and monitored individually. Hence, QoS parameters are identified specifically for each cloud layer. For example, certain QoS parameters can be targeted specifically in case of failures (e.g. DB component at PaaS layer). Consequently, CLAMS provides the means for more intelligent system scaling.

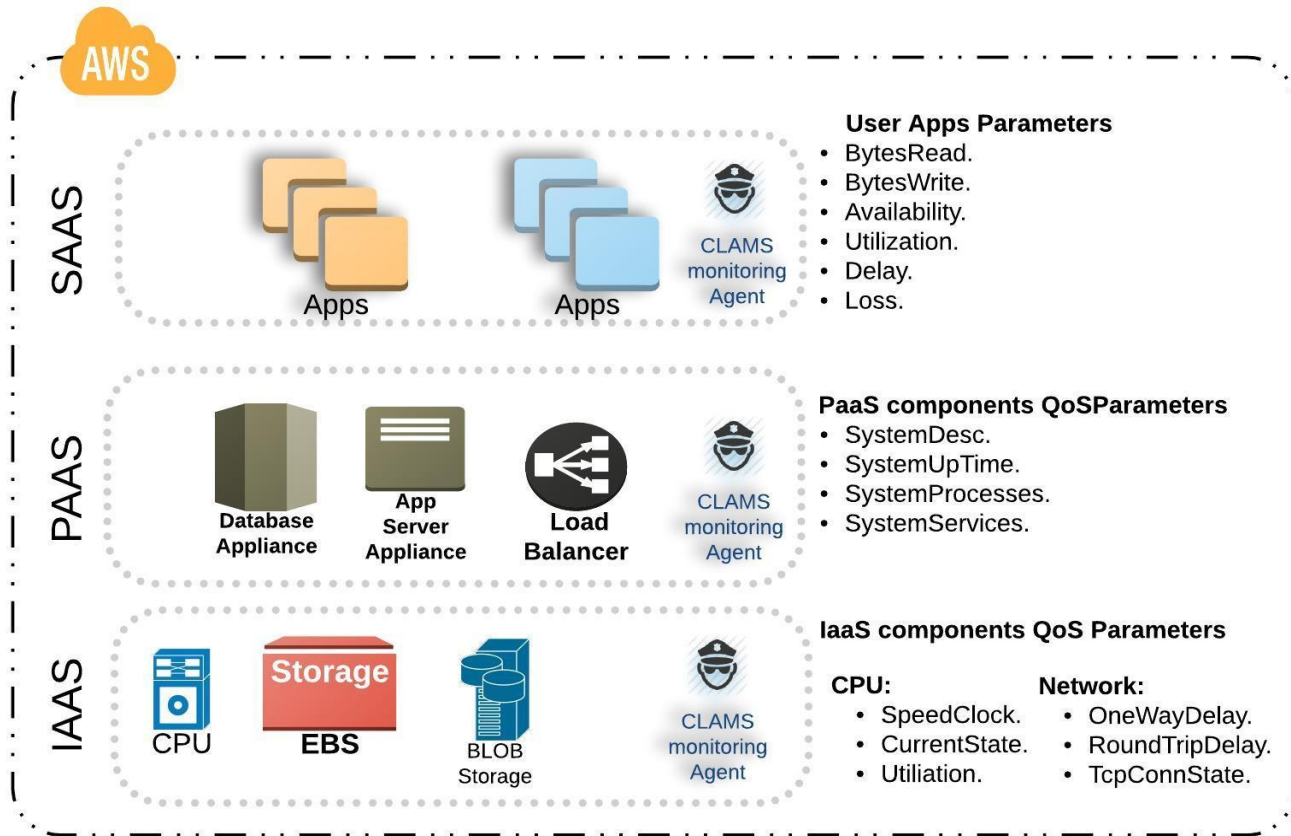


Figure 3.12: Applications components and QoS metrics cross-layers.

3.5.4. Using CLAMS to detect and identify at which cloud platform a failure occurs (Interoperability)

In situations where the ESA components are hosted across different regions by different cloud providers (such as monitoring issue 2, section 3.5.2), CLAMS can monitor all application stack components distributed individually independent of the cloud platforms. Therefore, system scaling can take the appropriate action eventually resulting in maintaining the agreed SLAs.

Previously, figure 3.12 demonstrated how the QoS metrics distributed across the cloud platform layers. Now, Figure 3.13, shows how different components are dis-

tributed across multi-cloud platforms. For illustration, the figure shows two different platforms (e.g. Amazon, and Azure) that have identical layers (e.g. SaaS, PaaS, and IaaS). Each layer of both platforms includes similar QoS performance parameters to be monitored. If using the current monitoring approaches, it is challenging to determine the origin of QoS parameter at which layer or on which cloud platform. Whereas, the CLAMS framework can determine the exact origin of each QoS parameter at which layer it operates and on which platform.

Eventually, the capabilities of CLAMS distributed Monitoring Agents enable the system administrator to gain the required QoS performance of these distributed components on multi-clouds. Therefore, the monitored system can take the appropriate auto-scaling actions ensuring the system performance level.

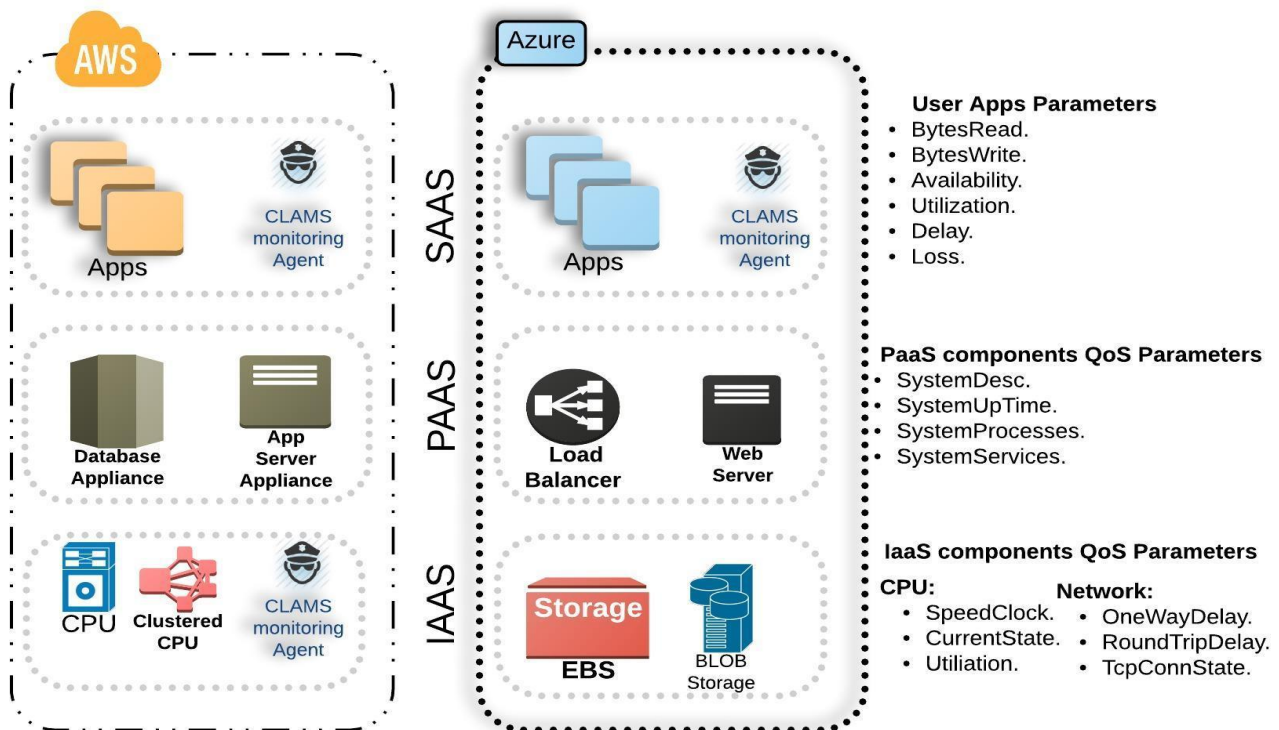


Figure 3.13: Applications components and QoS metrics across multi-cloud platforms.

3.5.5. CLALMS Data Collection Model Scenario

In section 3.2, we described the CLAMS data collection model. Table 3.2, presents how this data collection model is implemented for this scenario in order to achieve the visibility and interoperability of CLAMS monitoring. The table shows how CLAMS detects a specific QoS parameter for relative application's component and cloud platform layer. Moreover, figure 4.14 presents an ER relationship of a cloud platform provider, a specific Layer, a specific application's component, and a specific QoS parameter.

Cloud Layer	Layer Applications' Components	Targeted QoS Parameters
PaaS	Apache Server	Memory Utilization

Table 3-2: QoS parameters for specific resources across cloud platform layers.

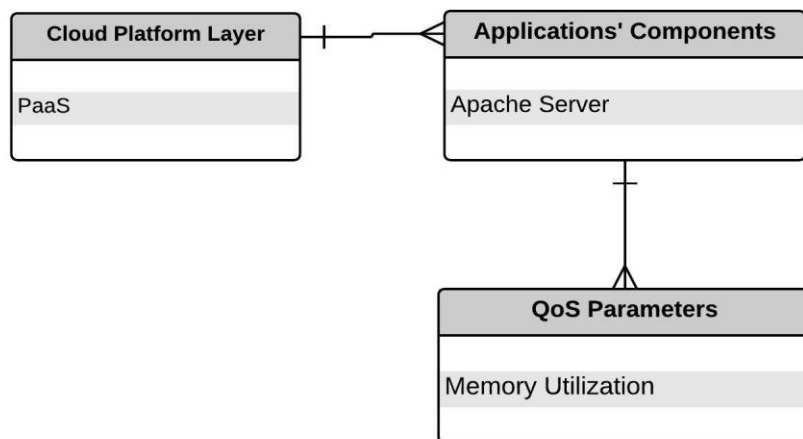


Figure 3.14: ER for the cloud layer, applications' components, and QoS parameters.

3.6. CLAMS vs. Other Monitoring Frameworks

In cloud environments, recent efforts have been put into improving VMs monitoring and controlling. A number of frameworks have been proposed for VM management, which include Simple Network Management Protocol (SNMP) for data retrieval. CloudCop is a conceptual network monitoring framework implemented using SNMP [120]. Basically, CloudCop adopts Service Oriented Enterprise (SOE) model. CloudCop 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 an application's components QoS performance monitoring. In addition to SNMP, CLAMS uses RESTful technology as an additional method for communication protocols.

Furthermore, there is a Management Information Base (MIB) called Virtual-Machines-MIB, that defines a standard interface for controlling and managing VM lifecycle [72]. It presents SNMP agents, which are developed based on NET-SNMP²⁹ 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 Libvirt³⁰ API and other resources such as VMM API are used in this framework [72]. However, Virtual-Machines-MIB is only concerned with monitoring IaaS-level (VM) QoS statistics. Unlike CLAMS framework, Virtual-Machines-MIB does not cater for the QoS statistics of PaaS level application components.

²⁹ <http://www.net-snmp.org/>

³⁰ <http://libvirt.org/>

Libvirt-snmp [72] is a project, which primarily provides SNMP functionality for libvirt. Libvirt-snmp allows monitoring of virtual domains; as well, it allows setting a domain's attributes. Furthermore, Libvirt-snmp provides a simple table containing monitored data about domain names, state, the number of CPUs, RAM, the RAM limit, and CPU time. In comparison to CLAMS, Libvirt-snmp does not allow monitoring an applications' components QoS performance across-layers.

Chapter 2 already discussed the properties of Cloudwatch on Amazon cloud platform and Azure FC on Microsoft Azure platform. Unlike CLAMS, both Cloudwatch and Azure FC do not allow cloud applications monitoring on multi-clouds. Furthermore, they do not enable the application's administrator to apply cross-layers monitoring, which CLAMS can do.

3.7. Summary

This chapter presented CLAMS—Cross-Layer Multi-Cloud Application Monitoring-as-a-Service Framework. The CLAMS framework aimed to break free of current black-box based cloud monitoring approaches [132] that give very little attention to individual components of applications provisioned across cloud layers. Figure 3.15, visualizes the features that CLAMS provides for an application's components across-layers on multi-cloud environments. The figure explains how an application's components could be distributed on different cloud platforms and also how the components are deployed across-layers. These abilities of CLAMS to monitor distributed applications' components make it different to current monitoring approaches.

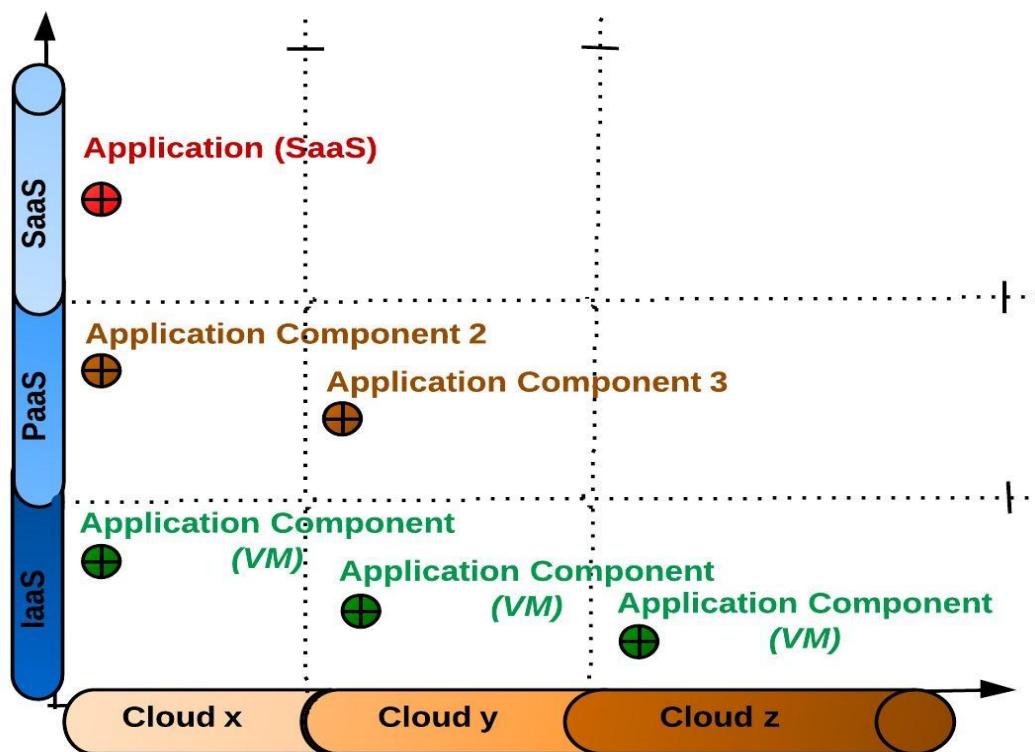


Figure 3.15: CLAMS – Cloud Monitoring Framework for cross-Layers applications components on multi-cloud Environments.

The novel features of CLAMS include the:

- (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 an application stack (e.g., web server, database server).

Chapter 4 will present (CLAMBS), which is an extension to CLAMS framework, which will incorporate benchmarking functionalities for cloud applications and re-

sources. Chapters 5 and 6 will present CLAMBS framework implementation using a number of different technologies and tools.

4. Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework

4.1. Introduction

This chapter proposes *CLAMBS: Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework*. CLAMBS represents an extension to CLAMS framework presented in chapter 3. Driven by the third research question (Chapter 1, Section 1.3), this extension enables benchmarking functionalities for cloud applications and resources based on the extensive study for monitoring and benchmarking design architecture presented in chapter 2. The proposed framework employs additional techniques to extend and enhance CLAMS framework in order to apply Cross-Layer Multi-Cloud Real-Time Application QoS Benchmarking.

In particular, CLAMBS offers the following novel features:

- It provides benchmarking-as-a-service that enables the establishment of baseline performance of an application deployed across multiple layers using a cloud-provider agnostic technique; and
- It is a comprehensive framework allowing continuous cross-layers benchmarking and monitoring on multi-clouds for hosted applications.

The chapter is organized as follows. Section 4.2 presents CLAMBS: Cross-Layer Multi-Cloud Application Monitoring as a Service. Section 4.3 presents CLAMBS architecture components. Section 4.4 presents the CLAMBS and the challenges of QoS and SLAs. Section 4.5 presents a comparison between CLAMBS framework and current monitoring and benchmarking frameworks. Section 4.6 concludes this chapter with a summary.

The Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking framework presented in this chapter has been published in the following paper (K Alhamazani et al., 2015).

4.2. CLAMBS: Cross-Layer Multi-Cloud Application Monitoring As a Service

This section describes the CLAMBS framework and its attributes as well as its different components. Figure 4.1 presents an overview of the proposed CLAMBS framework. As depicted in the figure, CLAMBS employs an agent based approach for cross-layers, multi-clouds resource/application monitoring and benchmarking. The CLAMBS framework is an extension of CLAMS framework by incorporating Benchmarking Agents. For the purpose of guaranteeing performance QoS and supporting the SLAs, the Benchmarking Agent adds new functionalities to CLAMBS as will be explained in this chapter. In this multi-cloud approach, Benchmarking Agents are deployed and distributed across various cloud provider environments based on application requirements and deployments.

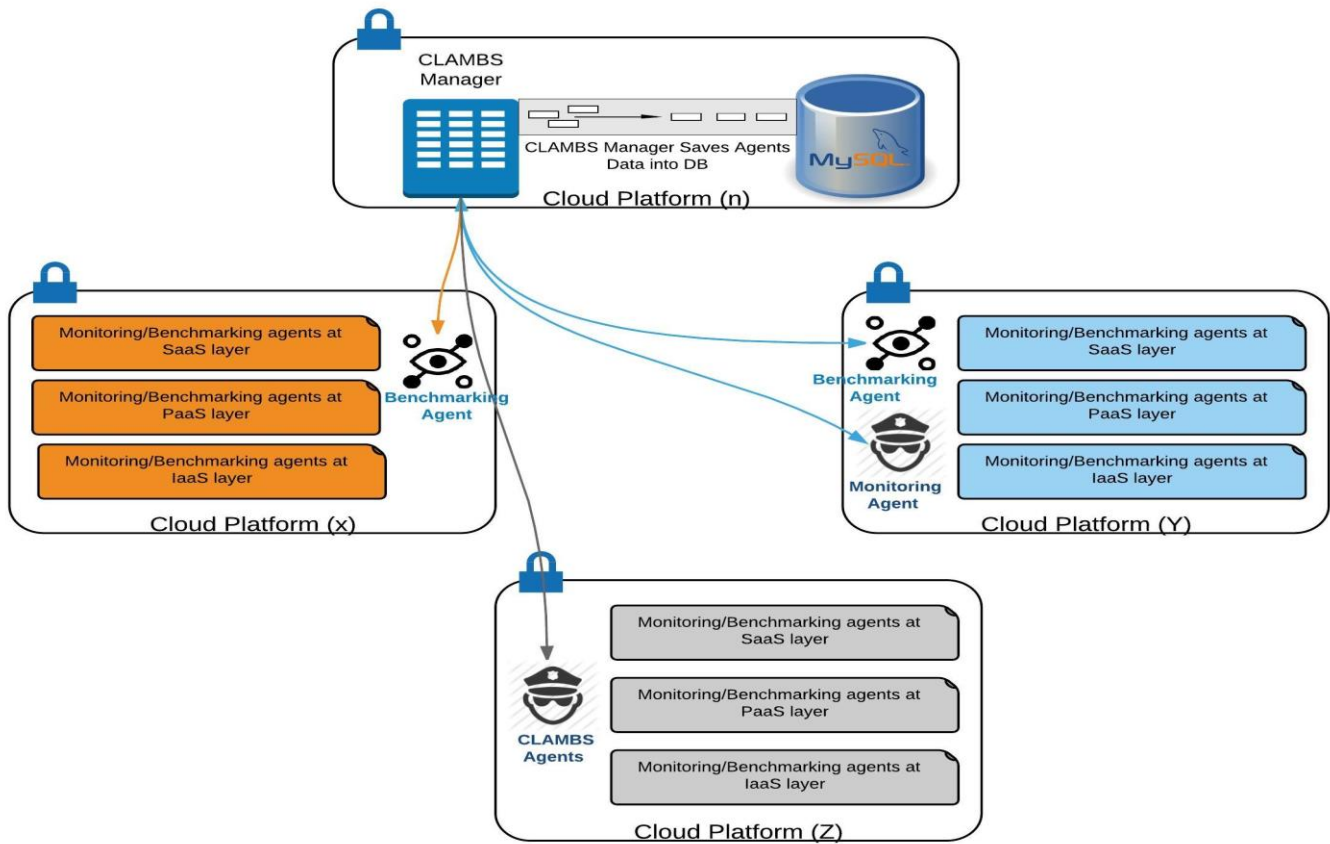


Figure 4.1: Overview of CLAMBS framework.

A CLAMBS Benchmarking Agent is responsible for benchmarking application QoS parameters such as resource consumption, network performance, storage performance etc., at various layers including SaaS, PaaS and IaaS. On the other hand, CLAMBS Benchmarking Manager is responsible for orchestrating and collecting QoS data from each Benchmarking Agent.

The CLAMBS Benchmarking Agents reside on various cloud layers, for example IaaS, PaaS, and SaaS. At each layer, one or more CLAMBS Benchmarking Agents can be assigned to benchmark hosted applications' components.

To illustrate, figure 4.2 describes how CLAMBS Benchmarking Agents are allocated on different various cloud platforms for monitoring applications' components at different cloud layers. The CLAMBS Benchmarking Manager and the CLAMBS database component can reside on the same cloud platform where CLAMBS Benchmarking Agents are running on different cloud platforms.

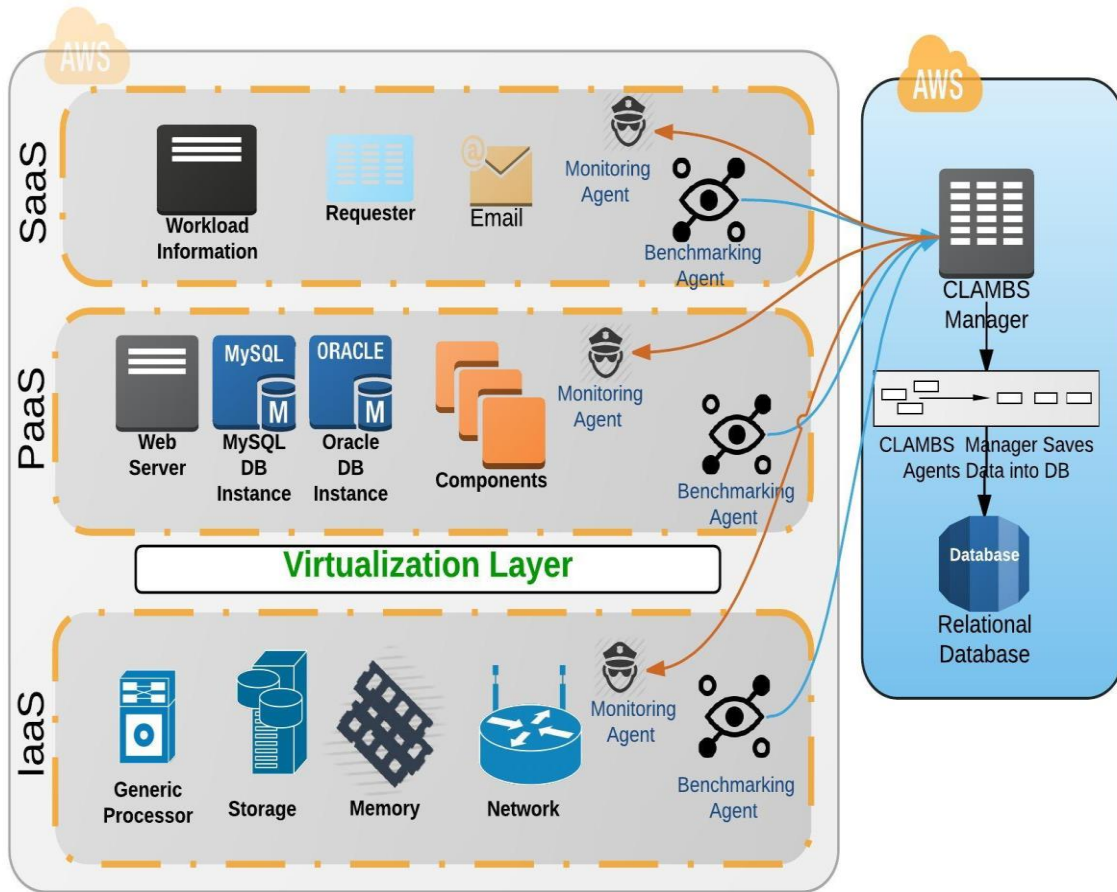


Figure 4.2: CLAMBS distributed architecture.

4.3. CLAMBS Architecture Components

CLAMBS includes mechanisms for efficient cloud monitoring and benchmarking applications deployed at *aaS layers. Furthermore, CLAMBS provides standard interfaces and communication protocols that enable the application/system administrator to gain awareness (benchmark and monitor against benchmarking outcomes) of the whole application stack across different cloud layers in heterogeneous envi-

environments (different resources constraints and operating systems). The CLAMBS approach also addresses the interoperability challenges among heterogeneous cloud providers. As shown in figure 4.3, the CLAMBS framework comprises two main components, namely, Benchmarking Manager and Benchmarking Agent.

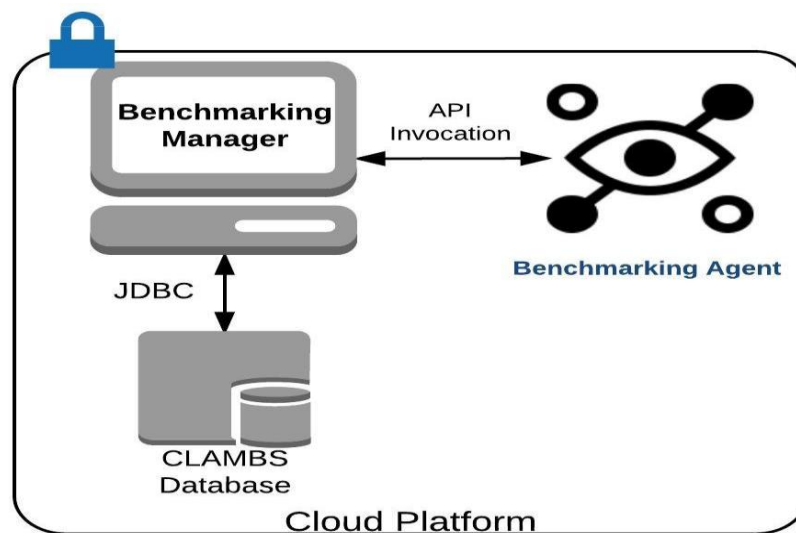


Figure 4.3: CLAMBS framework Benchmarking Manager and Benchmarking Agent components.

4.3.1. CLAMBS Architecture: Benchmarking Manager

The CLAMBS Benchmarking Manager is designed to facilitate benchmarking of applications components distributed across IaaS layers on multi-cloud environments.

The Manager's benchmarking function collects network and application performance QoS information from CLAMBS Benchmarking Agents that are distributed and running on several VMs hosted across multi-clouds environments in different

data centers. In particular, the Benchmarking Manager collects the traffic of QoS values from Benchmarking Agents hosted on VMs that are distributed across different datacenters. The Benchmarking Manager could be residing on the same cloud platform where Benchmarking Agents are running or it could be located on a different cloud platform.

The CLAMBS framework adopts the push/pull mode of communication as presented in section 3.3.1. Compared to the 'pull' mode, the main advantage of the 'push' mode is lowered Benchmarking Manager workload and faster response to errors. To handle the surge of report messages, the Benchmarking Manager employs a message queue to manage all incoming reports in a first-come-first-serve manner. Through analysis of a series of messages with consecutive time stamps (with sender agent ID), the Benchmarking Manager may also determine the Benchmarking Agents' work status whilst gathering QoS information for the cloud platform resources. Moreover, extending the above approach to a publish-subscribe paradigm is straightforward and can provide the ability to isolate errors based on priority and severity.

Furthermore, the Benchmarking Manager is responsible for firing VMs at remote data centers to perform application level benchmarking based on user requirements that include data center locations. For example, consider a scenario where an end user located in the Singapore datacenter, requests multimedia (audio/video) content from the audio/video streaming application service. Typically, such application components could be distributed across multiple datacenters (e.g. US Virginia, and AU Sydney). The CLAMBS framework supported by the Benchmarking Manager is able to dynamically fire a VM hosting the Benchmarking Agent at the end user location, (Singapore). Then the CLAMBS Benchmarking Manager can repeatedly test and benchmark the performance of the audio/video streaming application at both the locations (US Virginia, and AU Sydney) to select the best location to serve the streaming content to the end user.

This approach serves the following two main purposes:

- 1) it allows users who use third-party cloud hosting services to benchmark application performance for later comparison and evaluation; and
- 2) it allows users to test the system's performance automatically and choose the best performing datacenter for service delivery.

Here, the key advantage of CLAMBS is the ability to dynamically run benchmarking of applications at IaaS layers of multiple clouds automatically with very little configuration required from the user. The CLAMBS Benchmarking Manager also incorporates an API that is used by other Monitoring/Benchmarking Managers or external services to share the QoS statistics.

4.3.2. CLAMBS Architecture: Benchmarking Agent

The Benchmarking Agent is the second major component of the CLAMBS framework. Figure 4.1 presented the distributed CLAMBS Benchmarking Agents across the cloud layers. This Benchmarking Agent has the capability to migrate from the Manager VM to a VM that either hosts the application/service or acts as a client to the service. The rest of this section describes the features and capabilities of the CLAMBS Benchmarking Agent.

4.3.2.1 Workload Generator

The Benchmarking Agent incorporates standard functions to measure the network performance between the datacenter(s) hosting the application component and the client. The Benchmarking Agent incorporates a component that generates traffic to benchmark the application based on a workload model. This is able to generate load for different applications' components such as DBMS and Web Servers. For example,

it is able to generate requests to a web server (N users and M requests/second) based on a website workload model (e.g. football world cup trace - <http://ita.ee.lbl.gov/html/contrib/WorldCup.html>).

4.3.2.2 Capabilities

The Benchmarking Agent has the capability to work in multi-cloud heterogeneous environments. In essence, the objectives that require benchmarking process are:

- 1) Determining where and what type of performance improvements are needed.
- 2) Analyzing the available metrics of a performance.
- 3) Using benchmarking information in order to improve the services performance.
- 4) Comparing the benchmarking information with the standard measurements.

Thus, to benchmark cloud applications (e.g. a web application), providers can apply a generated workload on such application's distributed components.

4.4. CLAMBS and the Challenges of QoS and SLAs

4.4.1. CLAMBS for Unpredictable QoS parameters

In a cloud computing environment, the QoS parameter values are stochastic and can vary significantly based on unpredictable user workloads, hardware and software failures. This necessitates the awareness of cloud applications' current software and hardware resources status to meet the QoS targets of cloud-hosted applications. Cloud monitoring and benchmarking can assist in the holistic monitoring and awareness of applications and components at *aaS layers to meet agreed QoS in SLAs. Figure 4.4 describes how CLAMBS Benchmarking Agents are distributed across cloud layers on multi-cloud environments. This enables the CLAMBS to facilitate benchmarking cross-layers QoS parameters on multi-clouds.

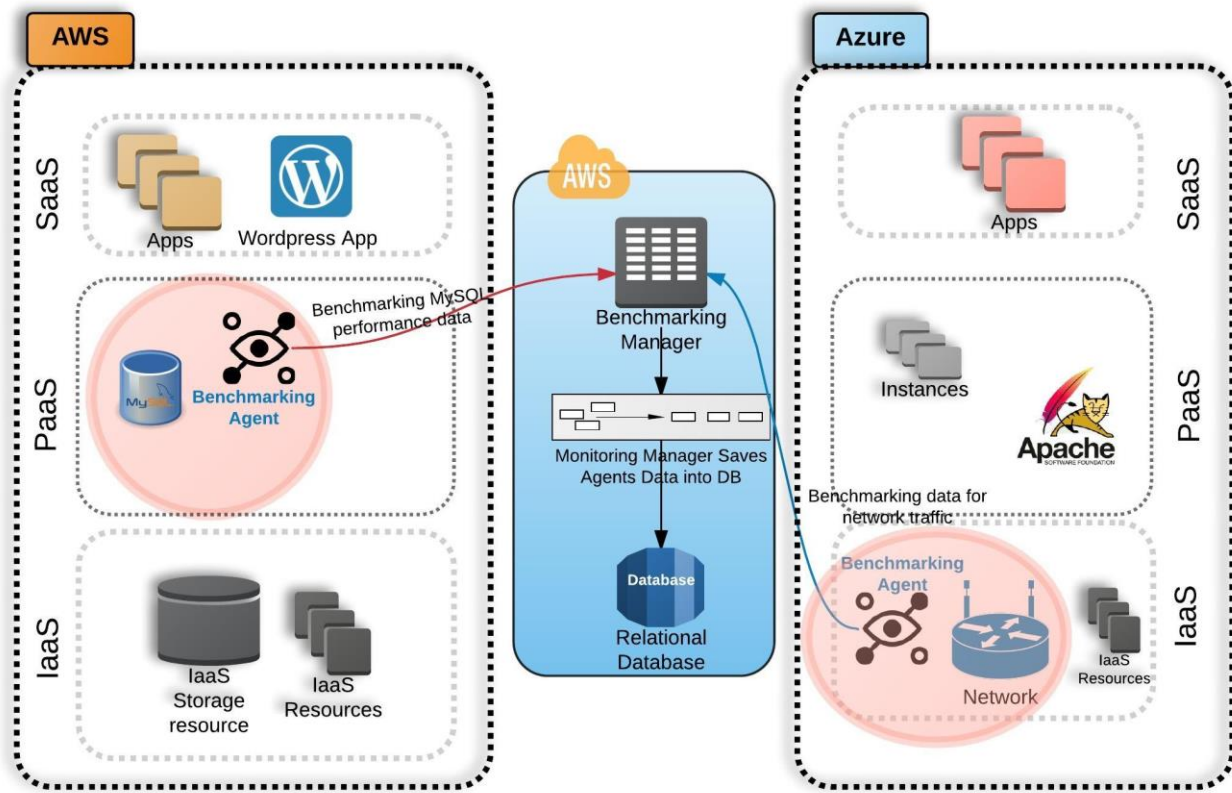


Figure 4.4: Visibility and interoperability of CLAMBS distributed components.

CLAMBS aimed to provide monitoring and benchmarking tasks for:

1. QoS management of software and hardware resources;
2. Runtime awareness of the applications and resources for cloud providers and application developers/administrators;
3. Detecting and debugging software and hardware problems affecting applications' QoS performance;
4. Understanding application performance (resource and network) before application deployment;

5. Facilitating application base lining; and
6. Enabling continual comparison of applications QoS performance against the targeted results.

4.4.2. CLAMBS for Addressing SLAs Challenges

A cloud platform is logically composed of three layers (SaaS, PaaS, and IaaS), applications such as email and games are hosted on SaaS layer; applications such as databases and web servers are hosted on the PaaS layer; and finally, IaaS include resources such as VMs, network and CPU resources. For the effective use of cloud resources and to meet SLAs, it is imperative that applications' components deployed across-layers on multi-clouds are monitored at runtime and benchmarked. In particular, application developers, system designers, engineers and administrators have to be aware of the compute, storage, networking resources, application performance and their respective quality of service (QoS) across all the cloud layers; this is because QoS parameters including latency and throughput play a critical role in upholding the grade of services delivered to the end customers based on the agreed upon SLAs.

To guarantee the SLA and to avoid failure, the challenge is to identify which component of the application needs to be re-configured or what type of auto-scaling is required. To this end, a better and accurate understanding of individual component's performance is needed to help a cloud orchestrator to effectively scale the corresponding layer at the appropriate time. The proposed CLAMBS framework benchmarking and real-time monitoring as-a-service system is a practical method to understand and evaluate how application components distributed across cloud layers on multi-cloud environments can essentially perform and handle their tasks.

4.5. CLAMBS vs. Benchmarking Frameworks

Chapter 2 explored traditional benchmarking approaches and elaborated why they cannot serve the users' needs. Besides runtime performance, cloud specific attributes such as elasticity, deployment, resiliency, and recovery are required to be reflected in the benchmarking process. Further, benchmarking applications distributed on multi-cloud environments is a complex task as each application requires evaluation of distinct QoS metrics from other QoS metrics in order to evaluate the targeted cloud performance. Moreover, each application has its own workload requirements for each individual component rendering the need for a general-purpose benchmark frame-

work. A specific purpose-built benchmarking component will not be able to serve cloud users having a variety of use cases in a cloud environment.

For web applications, a number of benchmarking frameworks have been designed. C-MART is the outcome of a notable effort to design a web application benchmarking tool. C-MART presents a significant tool, emulating, and then benchmarking web applications such as online stores or social networking websites. Originally, C-MART is motivated by the fact that benchmarks need to cope with the shift from the traditional environments to cloud environments. However, C-MART is limited to benchmarking web applications at the PaaS layer. Amazon EC2 compatible C-Meter was the original prototype of the EC2 current extensible cloud benchmark framework [179]. It employs low level metrics that are typically not visible to general cloud users. Therefore, C-Meter is unsuitable to evaluate higher levels of cloud services (e.g. PaaS and SaaS) [179]. In contrast, the CLAMBS Benchmarking Agents are not limited to benchmark web applications at the PaaS layer. This allows the CLAMBS Benchmarking Agents to benchmark different types of applications across layers.

Similarly, Compuware's Gomez [64], is a solution for web performance optimization (revenue based web and mobile applications). Gomez focuses on monitoring web applications from the end users' perspective. Moreover, it focuses on cross-browser testing and web load testing to optimize web-site performance. Furthermore, Gomez has released CloudSeuth as a web-based benchmark tool for the performance of Cloud providers. The main focus of ClouSeuth is availability (e.g. up time, downtime) [82], which measures the percentage of test transactions that completed successfully out of the set of transactions attempted. An unsuccessful test transaction is a transaction that returns a status code other than "200" in the maximum allowable timeframe [82]. However, Compuware lacks in its ability to monitor fine-grained application server and database events across multiple layers of the cloud. Whereas, the CLAMBS framework provides the ability to benchmark different applications components across the cloud platform. The CLAMBS Benchmarking Agents can be assigned to different components across the cloud layers, which provide a fine-grained monitoring.

An outstanding framework for cloud resources benchmarking is CloudCmp. For both the cloud provider and the cloud customer, the following is a list of the major features that CloudCmp can provide:

- 1) Provide performance and cost information about various cloud providers to a customer. The customer can use this information to select the right provider for its applications.
- 2) Enable the cloud provider to identify its under-performing services compared to its competitors.
- 3) Provide a comparison between different cloud providers by characterizing all providers using similar set of metrics. However, comparison is based on a specific pre-determined set of common cloud resources and services offered by all these cloud providers.

CloudCmp [96] chooses cost and performance metrics that are relevant to the typical cloud applications a customer deploys. Such metrics cover the main cloud services, including elastic computing, persistent storage, and intra-cloud and wide area networking. Nevertheless, some of the metrics provided by CloudCmp are too experimental to be meaningful to a cloud user, e.g. time to reach consistency. This limitation is not encountered with the CLAMBS framework, which provides meaningful QoS information to the cloud user. Moreover, CLAMBS Benchmarking Agents provide the application administrator with all the required explicit information for any targeted QoS parameter.

Additionally, efforts have been made for a virtualization layer and VMs management and benchmarking. CloudGauge presents an effective dynamic virtual ma-

chine benchmarking tool. It provides automated scripts to provision and measure the performance of the virtual environment setup. But, the focus of CloudGauge experimental benchmark was on the virtualization layer. Furthermore, the data collected was mainly CPU usage and average load Memory. Unlike CloudGauge, the CLAMBS Benchmarking Agents are not restricted to test only CPU usage or average load memory; rather it has the ability to retrieve various QoS parameters. That is, a CLAMBS Benchmarking Agent is not only restricted to a number of performance QoS targets.

Furthermore, COSBench is a benchmarking tool for characterizing object storage services, thus allowing people to evaluate various implementations or configurations of object storage service. Basically, Object storage services provide RESTful interfaces to store and access files in a way that is similar to regular file systems but in a simpler method. Thus, COSBench allows users to evaluate a number of implementations or configurations of object storage service. However, COSBench is only capable of measuring mainly three performance metrics (e.g. Response Time, Throughput, and Bandwidth). This limitation is overwhelmed in the CLAMBS framework, which is not restricted to benchmarking a certain number of QoS parameters. Furthermore, COSBench currently has adaptors for only Amazon S3 and Rackspace

Cloud Files, which restricts the interoperability that CLAMBS framework can offer. CLAMBS Benchmarking Manager and Agents are agnostic to cloud platforms.

In addition to above the benchmarking frameworks, the mOSAIC benchmarking tool is designed based on the mOSAIC platform, which provides a simple way to develop cloud applications [133, 134]. In the mOSAIC, a cloud application is structured as a set of components running on cloud resources (i.e., on resources leased by a cloud provider) and able to communicate with each other. Therefore, the main users for the mOSAIC benchmarking tool are the mOSAIC developer and the mOSAIC application final user. In other words, mOSAIC benchmarking tools acquire the mOSAIC framework to run on different cloud platforms. This restriction does not exist when using the CLAMBS framework for cloud application benchmarking. CLAMBS components (e.g. CLAMBS Manager/Agents) are agnostic to any specific platform or framework.

Still, most of the current benchmarking interfaces are different among VM platforms. Hence, there is a necessity for benchmarking frameworks that adopt unified interface for multiple virtualization platforms, which CLAMBS framework adopts.

4.6. Summary

This chapter presented CLAMBS—Cross-Layer Multi-Cloud Application Monitoring and Benchmarking-as-a-Service Framework. The novel features of CLAMBS includes:

- (i) The distributed CLAMBS Benchmarking Agents (shown in figure 4.2) provide benchmarking-as-a-service that enables the establishment of baseline performance of application deployed across multiple layers using a cloud-provider agnostic technique; and
- (ii) It is a comprehensive framework allowing continuous benchmarking and monitoring of multi-clouds, multi-layers hosted applications.

Chapters 5 and 6 will present how the CLAMBS framework is implemented using a number of different technologies and tools. Moreover, they will show a number of experimental results and outcomes to validate CLAMBS framework.

5. Modelling and Implementation of CLAMS and CLAMBS Framework

5.1. Introduction

Chapter 3 presented the proposed cloud applications monitoring framework, namely, CLAMS framework that enables cloud users to monitor applications' components across-layers on multi-clouds. Chapter 4 presented the extension of the proposed architecture to add benchmarking functionalities in order to benchmark cloud resources and applications' components, namely, the CLAMBS framework.

This chapter presents a prototype implementation of the CLAMBS framework. The prototype implementation is divided into two parts:

- Determining architecture and implementation technologies, (Section 5.2).
- Determining the optimal deployment on cloud platform, (Section 5.3).

This chapter is organized as follows. Section 5.2 presents and discusses the proof-of-concept of the implementation of the CLAMBS framework. Also, it explores the development tools and prototype environment utilized for the implementation. Sec-

tion 5.3 presents modeling and analyzing CLAMBS overheads in Multi-Cloud Environments. Section 5.4 concludes this chapter with a summary.

The implementation, modelling and analyzing of the proposed framework presented in this chapter has been published in the following papers (K Alhamazani et al., 2014b, K Alhamazani et al., 2014c, K Alhamazani et al., 2015).

5.2. Proof-of-Concept Implementation

The proposed framework does not rely on other frameworks or specialized hardware. CLAMS and CLAMBS are both generic frameworks that are not restricted by any hardware or software specifications.

5.2.1. Development Tools and Techniques

To begin with, this sub-section presents an overview of the development tools utilized for implementing the proposed framework prototype. Furthermore, it will provide details of the environment and cloud platforms that were used to process the implementation throughout the framework progression.

5.2.1.1 JAVA

Formally, Java was first announced by Microsystems on 1995 [149]. The Java language is both a conventional and rapid prototyping language. Java is a simple, object oriented, distributed, interpreted, robust, secure, platform independent, portable, high-performance, multi-threaded and dynamic language. Moreover, the Java Virtual Machine (JVM) represents the Java run-time environment. Therefore, Java is an architecture-neutral, or platform-independent, or multi-platform language. Java enables web browsers to automatically download Java applets across the network. These Java applets will be executed within the JVM of the local machine, which releases the CPU of the remote machine that hosts the applets. Since Java is designed to be used on the Internet, it comprises several networking libraries such as Transmission Control Protocol/ Internet Protocol (TCP/IP) networking.

5.2.1.2 Eclipse

Eclipse is an integrated development environment (IDE) for Java development. It encompasses a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and used for developing Java applications. Moreover, the Eclipse platform defines an open architecture so that each plug-in development team can focus on their area of expertise [53].

Embedded plug-ins in Eclipse can also be used to develop applications in other programming languages (e.g. Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Lua, Natural, Perl, PHP, Prolog, Python, R, Ruby, Scala, Clojure, Groovy, Scheme, and Erlang).

5.2.1.3 Apache Tomcat

The Apache HTTP Server is a robust software development, which is a free open source implementation of the HTTP web server [13]. Originally, the Apache project is part of the Apache Software Foundation. In addition, a huge number of users have contributed ideas, code, and documentation to the Apache project. Apache Software provides robust and commercial-grade reference implementations of many types of software. Hence, Apache Tomcat is a readily available and reachable platform which can be utilized by developers and organizations to implement systems for both experimental and other targets.

5.2.1.4 Simple Network Management Protocol (SNMP)

5.2.1.4.1 Overview

SNMP is a protocol that enables servers to share information about each other's current condition, and also a channel through which a network administrator can modify pre-defined values to specify the targeted shared information. While the SNMP protocol itself is very simple, the structure of programs that implement SNMP can be very complex [125].

In concept, SNMP is a protocol that is implemented on the application layer of the networking stack. Basically, SNMP was designed as a method of collecting information from very different systems in a consistent manner. Even though it can be used in connection to a diverse array of systems, the method of enquiring data and the paths to the relevant information are standardized.

Originally, SNMP was first introduced in the late 1980s [153]. Today, there are multiple versions of the SNMP protocol, and many networked hardware devices implement some form of SNMP access. The most widely used version is SNMPv1, but it is in many ways insecure. The enhanced version is SNMPv3, which provides more advanced security features.

5.2.1.4.2 SNMP MIBs

The Management Information Base (MIB) is a database to manage the entities in networks. MIB is a hierarchical structure that, in many areas, is universally standardized, but also flexible enough to allow vendor-specific additions. An MIB structure can be described as a top-down tree where each branch that forks is labeled with both an identifying number (starting with 1) and an identifying string that are unique for that level of the hierarchy. To refer to a specific node of the tree, you must trace the path from the unnamed root of the tree to the targeted node. The lineage of its parent IDs (numbers or strings) are strung together, starting with the most general, to form an address. Each junction in the hierarchy is represented by a dot in this notation, so that the address ends up being a series of ID strings or numbers separated by dots. The entire address is known as an Object Identifier (OID).

5.2.1.4.3 SNMP4J

SNMP4J is the object oriented SNMP API for developing Java managers and agents. SNMP4J is an enterprise class, free open source, and state-of-the-art SNMP v1/2c/v3 implementation using Java. SNMP4J is the core API for implementing any SNMP service. Currently there is a lack of an affordable object oriented designed SNMP implementation for Java. SNMP4J tries to fill this gap; hence, it is free to get the best support and feedback from the Internet community.

5.2.1.5 SIGAR (System Information Gatherer and Reporter)

SIGAR (System Information Gatherer and Reporter) was designed and implemented by Doug MacEachern at Covalent Technologies starting in September of 2002 and has continued with Hyperic as a core component of the HQ product [75]. SIGAR is a cross-platform, cross-language library and command-line tool for accessing operating system and hardware level information in Java, Perl and .Net [141]. For illustration, SIGAR APIs provide users to gather information such as [75] [141]:

- System memory, swap, cpu, load average, uptime, logins
- Per-process memory, cpu, credential info, state, arguments, environment, open files
- File system detection and metrics
- Network interface detection, configuration info and metrics
- TCP and UDP connection tables
- Network route table

5.2.1.6 Restlet

Restlet is a lightweight, comprehensive, open source RESTful web API framework for the Java platform. Restlet is suitable for both server and client Web applications. It supports major Internet transport, data format, and service description standards like HTTP and HTTPS, SMTP, XML, JSON, Atom, and WADL. A GWT port of the

client-side library is also available, as well as other editions for Android, OSGi and Google App Engine.

5.2.1.7 JMeter

JMeter, which is implemented by Apache organization is an open source pressure test tool based on Java. JMeter can be used on servers, networks or other objects to simulate huge loading, and testing their strength and analyzing the overall performance under different variety of pressure [172]. Originally, Apache JMeter was designed and tested for Web Applications. Later on, it was extended to test other functions. In addition, Apache JMeter can test performance on static and dynamic resources [10]. Moreover, JMeter can generate reports of resulted data for the user.

5.2.2. Cloud Platforms Used

This section presents the environments on which the prototype is executed. For the proof of concept objectives, the prototype was executed onto two major public cloud platforms. Although, the prototype has no limitation and can be executed on any other cloud platform. Thus, the main reason for our choice of platforms was made due to their publicity and the offered services for new registered accounts.

5.2.2.1 Amazon Web Services (AWS) Elastic Computing (EC2)

Amazon Web Services (AWS) is a broad, evolving cloud computing platform provided by Amazon. Historically, the first AWS offerings were launched in 2006 to provide online services for websites and client-side applications [7]. AWS is geographically spread over several regions. These regions have central hubs in the Eastern USA, Western USA (two locations), Brazil, Ireland, Singapore, Japan, and Australia. Each region includes multiple smaller geographic areas called availability zones.

5.2.2.2 Microsoft Windows Azure Platform

Azure platform is provided by Microsoft. for building, deploying and managing applications and services through a global network of Microsoft managed, and Microsoft partner hosted, datacenters [110]. It was announced in October 2008 and released on 1 February 2010 as Windows Azure, before being renamed to Microsoft Azure on 25 March 2014.

5.2.3. CLAMBS: A Practical System Prototype

This section presents how the full version of the CLAMBS prototype has been implemented. The prototype has been developed using Java and is completely cross-

platform interoperable; that is, it works on both Windows and/or Linux operating systems. Figure 5.1 shows the framework components deployment using a UML diagram.

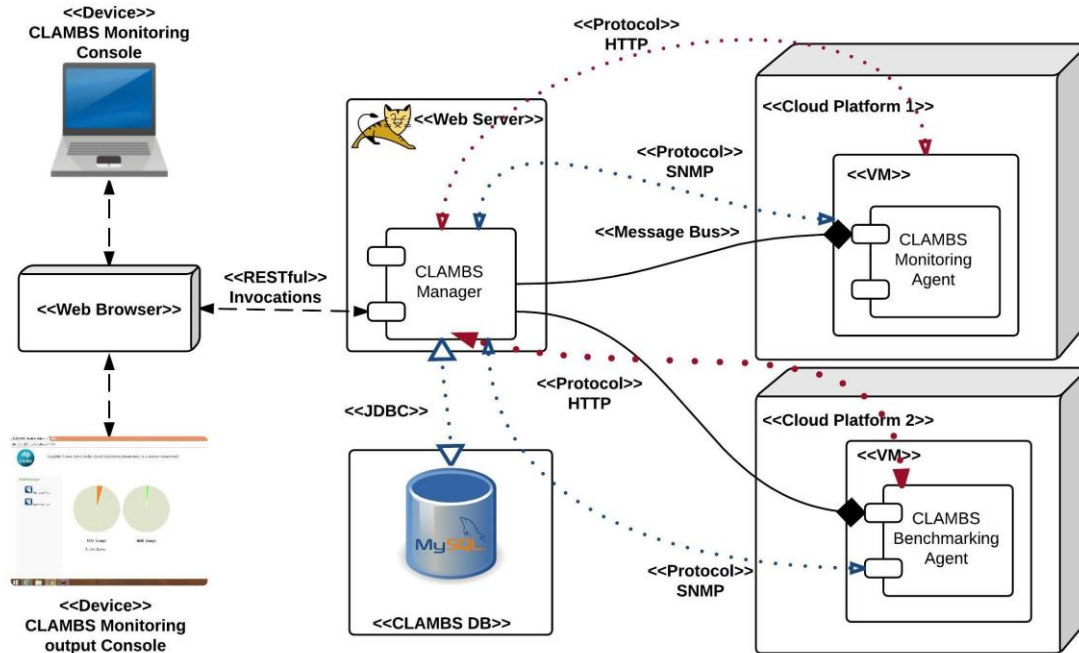


Figure 5.1: UML based description of the CLAMBS framework.

As illustrated in chapters 3 and 4, the CLAMBS Manager component is connecting to CLAMBS Monitoring Agents and Benchmarking Agents hosted on a cloud platform. For the first version (CLAMS) the main protocol used for communication is SNMP, as shown in the figure. Whereas, in the extended version (CLAMBS) the main protocol used for communication is through RESTful APIs (HTTP). Moreover, the CLAMBS Manager profiles the data retrieved from CLAMBS Agents into

CLAMBS DB, which is MySQL DBMS. Moreover, the web server used for the CLAMBS components to run and communicate is Tomcat Apache, as depicted in the figure.

Figure 5.2 presents a screenshot of the CLAMBS monitoring console. The figure shows how distributed CLAMBS Agents run and retrieve data from two different platforms (see figure 4.4), namely, the Azure platform and the Amazon EC2 platform. The output in the screenshot demonstrates the values of targeted QoS of an application's component and the values relative to the CLAMBS Agent itself, e.g. the name of the CLAMBS Agent, the location of the CLAMBS Agent, and utilized memory by the application's component.

```

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 5.2: CLAMBS proof-of-concept Implementation.

Table 5.1 presents more illustration of the output of CLAMBS monitoring console shown in figure 5.2. The table lists out some of the presented output and the meaning of each output line.

Output	Meaning
AzureAgent	Name of the CLAMBS Agent.
AzureAgent. AzureAgent.il.internal.cloudapp.net	The CLAMBS Agent location domain.
536	The monitored component ID.
taskhostex	The monitored component name.
64.74	Memory utilization by the monitored VM.

Table 5-1: Illustration of the CLAMBS monitoring console output.

5.2.3.1 CLAMBS Monitoring Agent Implementation

The process of retrieving QoS targets (section 3.3) is done by utilizing functionalities provided by SNMP, SIGAR³¹ and other custom built APIs. For instance, SNMP is used in CLAMBS Monitoring/Benchmarking Manager to retrieve the QoS values related to networking, the number of packets in and out, the route information and, the number of network interfaces. SIGAR is also used in CLAMBS Monitoring/Benchmarking Manager to obtain access to low-level system information such as

³¹ <http://www.hyperic.com/products/sigar>

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 are also obtained using SIGAR in CLAMBS Monitoring/Benchmarking Agents. 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, new SNMP Objects Identifiers (OIDs) are identified in a sequence within both CLAMBS Monitoring/Benchmarking Manager and Agents. For example, the 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, the function to get process memory is assigned an OID .1.3.6.1.9.1.1.0.1. Figure 5.3, shows a code snippet for Monitoring Agent implementing SIGAR functionalities. Similarly, figure 5.4 shows the code snippet for implementing SNMP in the Monitoring Agent.

```

@Get
public String getTargetInfo() {
    try {
        info= new AgentInfo();
        sigarHelper = new SigarHelper();

        // get pid by Process name
        long pid = SigarHelper.getProcessId(getProcessName());
        // get pid info by pid
        info.getProcInfo(Long.toString(pid));

        hostname = info.getHostName();
        IP_addr = info.getAddress();
        Proc_id=info.Proc_id;
        Proc_stat=info.Proc_stat;
        Proc_mem=info.Proc_mem;

        org.hyperic.sigar.CpuInfo[] infos = sigar.getCpuInfoList();
        //Get the utilized CPU
        CpuPerc[] cpus = sigar.getCpuPercList();
        org.hyperic.sigar.CpuInfo cpuInfo = infos[0];
        DecimalFormatSymbols simbol = new DecimalFormatSymbols();
        simbol.setDecimalSeparator('.');
        DecimalFormat df1 = new DecimalFormat("##.##",simbol);
        Proc_cpu = df1.format(sigar.getCpuPerc().getUser()*100);
        //Get the utilized memory
        mem = sigar.getMem();
        actualFreeMem = " "+mem.getActualFree() / 1024 / 1024;
        actualUsedMem = " "+mem.getActualUsed() / 1024 / 1024;
        totalRAM = " "+mem.getRam();

    } catch (SigarException se) {
        se.printStackTrace();
    }
}

```

Figure 5.3: SIGAR CLAMBSMonitoringAgent.java – Code Snippet.

```

protected void registerMIBs()
{
    if(modules == null)
        modules=new MyModules(getFactory());
    try {

        MScalar myScalar0 = new MScalar(new
OID("1.3.6.1.9.1.1.0.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.hostname));
        MScalar myScalar1 = new MScalar(new
OID("1.3.6.1.9.1.1.1.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.IP_addr));
        MScalar myScalar2 = new MScalar(new
OID("1.3.6.1.9.1.1.2.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.Proc_id));
        MScalar myScalar3 = new MScalar(new
OID("1.3.6.1.9.1.1.3.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.Proc_stat));
        MScalar myScalar4 = new MScalar(new
OID("1.3.6.1.9.1.1.4.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.Proc_mem));
        MScalar myScalar5 = new MScalar(new
OID("1.3.6.1.9.1.1.5.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.Proc_cpu));
        MScalar myScalar6 = new MScalar(new
OID("1.3.6.1.9.1.1.6.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.actualFreeMem));
        MScalar myScalar7 = new MScalar(new
OID("1.3.6.1.9.1.1.7.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.actualUsedMem));
        MScalar myScalar8 = new MScalar(new
OID("1.3.6.1.9.1.1.8.0"),MOAccessImpl.ACCESS_READ_CREATE,new
OctetString(this.totalRAM));

    }
    catch (DuplicateRegistrationException drex) {
        drex.printStackTrace();
    }
}

```

Figure 5.4: SNMP CLAMBSMonitoringAgent.java – Code Snippet.

Furthermore, the CLAMBS implementation also incorporates an HTTP based Restful communication standard between Monitoring/Benchmarking Manager and Agents.

This allows greater flexibility to monitor an application that does not support the network specific SNMP protocol.

5.2.3.2 CLAMBS Benchmarking Agent Implementation

Section 4.3 presented the CLAMBS Benchmarking Agent component. Benchmarking Agents are bootstrapped with the VMs and distributed across different cloud platforms e.g. Amazon and Azure. On booting VMs, CLAMBS Benchmarking Agents start up and wait for incoming requests from the CLAMBS Manager to start benchmarking. Typically, there is a unique IP address for each Agent representing the VM location. The port used for communication by the Benchmarking Agents is 80, as the protocol identifier in our implementation for communication is HTTP. The server component integrated to run the CLAMBS Agents is Apache Tomcat. Upon requests by the CLAMBS Manager, the CLAMBS Agent starts its role, which includes download/upload objects from remote server. Essentially, the CLAMBS Agent is capable of handling requests from more than one CLAMBS Manager in case a hierarchical architecture is adopted (see section 3.3.3). The Benchmarking Agent also incorporates the load generator functionalities. This load generator function of CLAMBS is essentially implemented using the JMeter package developed in Java. In this implementation, the prototype is designed to generate web application server traffic using HTTP requests. The load generator in CLAMBS framework also supports SQL load genera-

tion. In the case of HTTP workload, HTTP sampler is provided along with the domain, port number, path, and the request method (e.g. POST or GET). Similarly, in the case of the SQL workload, SQL sampler, query, query type (insert, update, or select), database URL, and database driver are provided. A loop controller is specified according to the aimed workload scenario. This also applies to the thread group and the number of threads that will perform the intended workload. Seamlessly, the CLAMBS load generator prototype is implemented to be able to reach the targeted components across different cloud platforms.

5.2.3.3 CLAMBS Manager Implementation

The CLAMBS Manager was presented in section 3.3.1 and section 4.3.1. The CLAMBS Manager uses MySQL database to store the QoS statistics collected from the Monitoring and Benchmarking Agents. For the proof-of-concept implementation, a pull approach is used when the CLAMBS Manager is responsible to poll for QoS data from CLAMBS Agents distributed across multiple cloud provider VMs. The CLAMBS Manager uses a simple broadcasting mechanism for CLAMBS Agent discovery. On booting, a discovery message is broadcasted to the known networks. CLAMBS Agents that are available respond to the Manager's request. The CLAMBS Manager then records CLAMBS Agent information to the Agent database. The CLAMBS Manager then starts off threads to query each CLAMBS Agent in the

Agent database to obtain QoS parameters. Furthermore, the discovery method can be used while run-time for recovery purposes when one Agent stops sending data or fails. The polling interval is a pre-defined constant and can be changed using the CLAMBS Manager configuration files.

Utilizing Java functionalities, the CLAMBS Manager is implemented based on the net package provided by Java libraries. This library is responsible for most network communication functions and requirements. It provides the superclass `URLConnection`, which represents a communication link between applications and Uniform Resource Locator (URL). Therefore, each CLAMBS Manager's request will have two main components: the protocol identifier and resource name. The benchmarking component of the CLAMBS Manager can measure the QoS parameters including Network Latency, Network Bandwidth, Network download speed, and Network upload speed.

The CLAMBS Manager implements a RESTful API allowing other applications to request QoS data of individual application components running in a multi-cloud environment. The web interface consumes the API to present the data about applications and Agents visually. The web interface is developed using HTML5 and JavaScript. Figure 5.5, and 5.6 present the web interface screenshots.

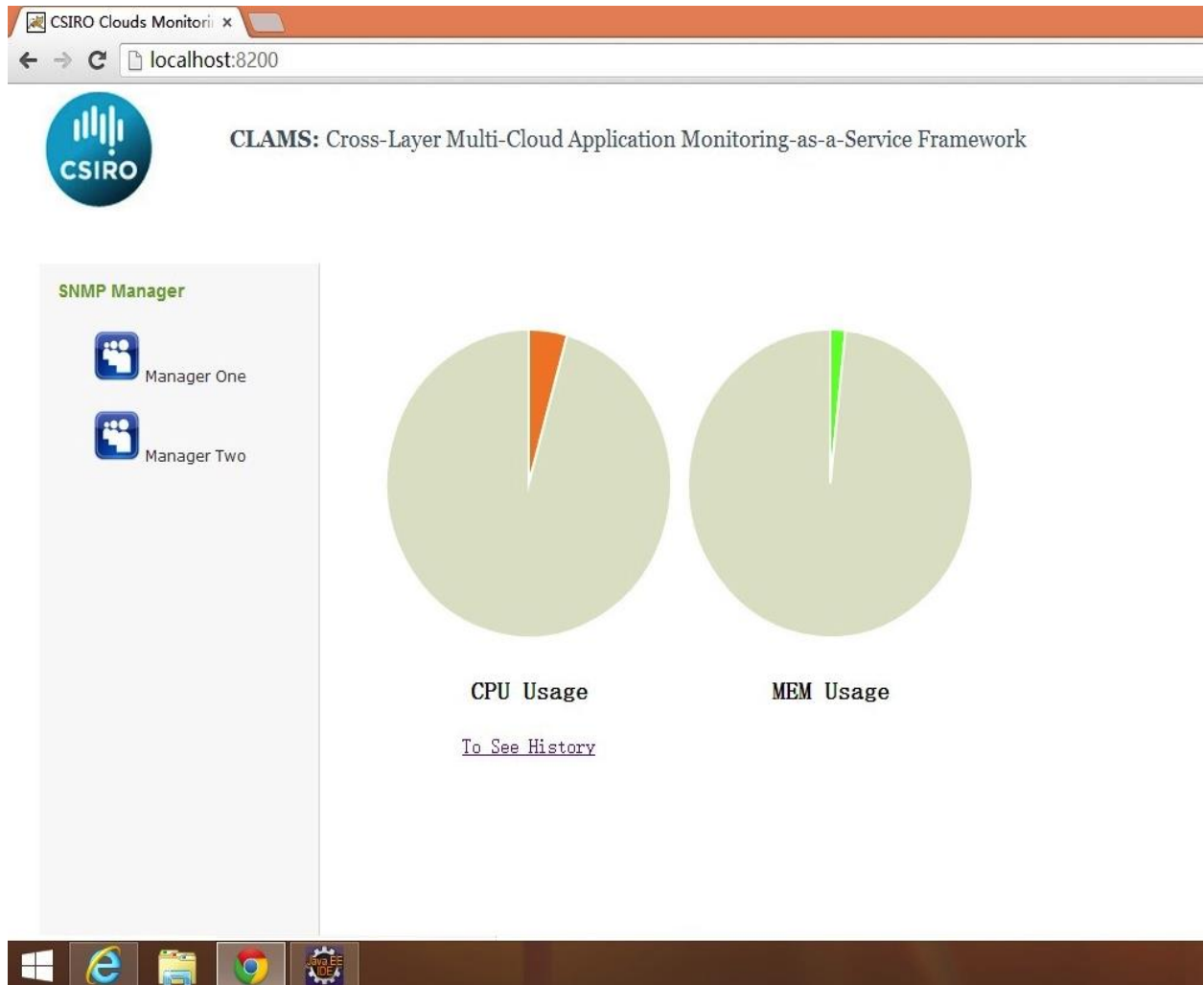


Figure 5.5: Screenshots of CPU and MEM Usage.

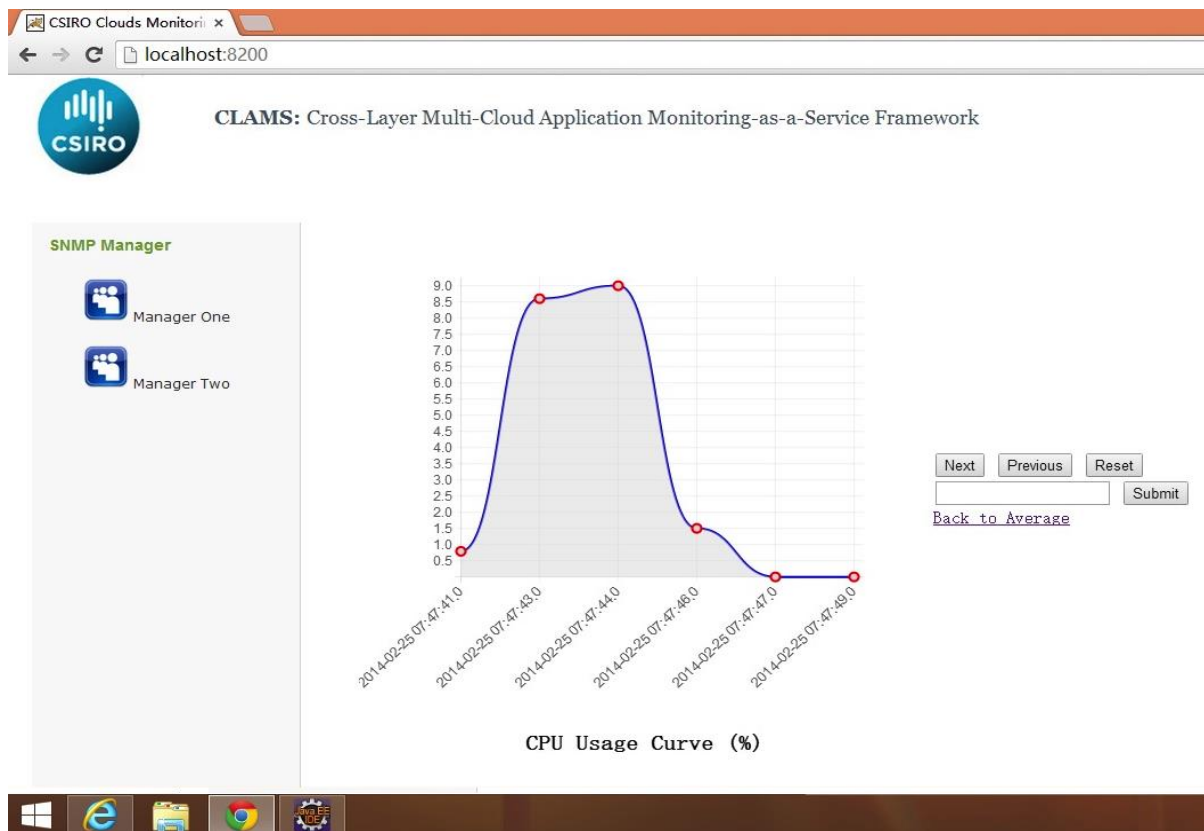


Figure 5.6: Screenshots of the Curve of CPU Usage.

5.2.3.4 CLAMBS Agent and Manager Communication

For the proof-of-concept implementation, the communication between the CLAMBS 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 CLAMBS Agents and Manager/Super-Manager. Using a standardized SNMP interface makes CLAMBS completely compatible with existing SNMP-based applications, tools and systems, and reduces the effort involved in col-

lecting QoS statistics. Figure 5.7 shows a snippet code for assigning a unique port number for each CLAMBS Monitoring Agent.

```
for (int i = 8001; i<8002; i++)  
{  
    agentName = " "+i;  
    agent.setName(agentName);  
    agent = new Server(Protocol.HTTP,i,AgentResource.class);  
  
    agent.start();  
}
```

Figure 5.7: Assigning unique port number for each CLAMBS Monitoring Agent – Snippet Code.

Furthermore, figure 5.8 demonstrates the communication flow based on multiple protocols between the CLAMBS Manager and Agents.

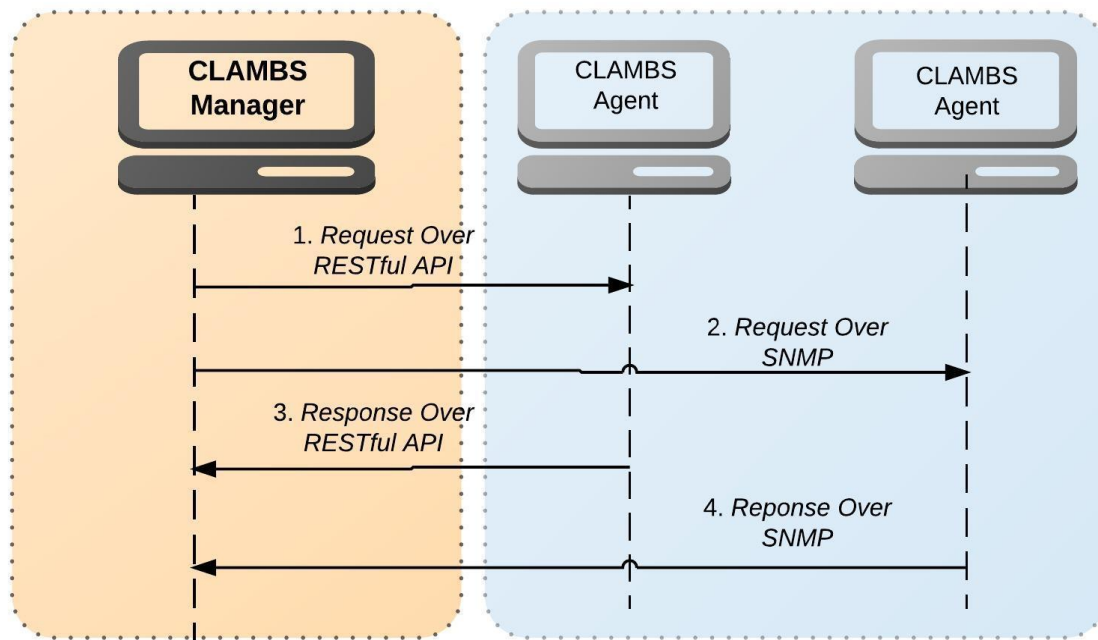


Figure 5.8: CLAMBS components communication based on RESTful and SNMP.

5.3. Modeling and analyzing CLAMBS Overheads in Multi-Cloud Environments

The abstract model and analysis of CLAMBS framework will enable:

- Evaluating the possible overheads of the CLAMBS framework.
- Evaluating the deployments model of the CLAMBS framework.
- Choosing the deployment model for the CLAMBS framework.

The pervious section presented the CLAMBS prototype implementation using various technologies. However, the deployment of the implemented prototype can be done in multiple ways of deployment, mainly centralized or decentralized. This section will firstly present an abstract deployment model of the CLAMBS prototype in section 5.3.1. Moreover, this section will present a study of the overheads of the CLAMBS framework. The overheads study will enable deciding which deployment model to adopt to conduct the experiments (presented in chapter 6).

The CLAMBS framework is agnostic of the underlying cloud platform; that is, the CLAMBS Manager/Agent may run on heterogeneous cloud platforms (chapter 3). In case the CLAMBS framework components are distributed across different cloud platforms (e.g. Amazon cloud platform and Windows Azure platform), one CLAMBS Manager and multiple CLAMBS Agents will be residing on each of these cloud platforms. Hence, it is important to model the overheads introduced by the distribution of CLAMBS in multi-cloud environments. As there are different performance requirements for different cloud applications, the CLAMBS framework must cope with the specific requirements with different deployments.

5.3.1. Abstract Model for CLAMBS framework Deployment

This section introduces an abstract model for the CLAMBS framework and analyzes the performance of the CLAMBS framework itself. Table 5.2 lists all notation for this analysis model.

Parameter	Description
Cloud Infrastructure	
$A = \{a1, \dots a_m\}$	Set of applications.
$C = \{c1, \dots c_i\}$	Set of application components.
$V = \{v1, \dots v_m\}$	Set of m cloud VM images.
$S = \{s1, \dots S_n\}$	Set of n cloud infrastructure services.
$P = \{p1, \dots p_o\}$	Set of o cloud providers.
$D = \{D1, \dots D_n\}$	Set of Datacentres.
$L = \{L1, \dots Li\}$	Set of Datacentres' Locations.
Applications	
α	Location of an application component.
β	Location of a PaaS component e.g. VM.
φ	Location of Agent.
ζ	Location of CLAMS user (CLAMS Manager).
ψ	Set of QoS Parameters of Application.
Ω	Set of QoS Parameters of PaaS level component e.g. (Network, Storage, and VM).
G	Application workload.
CLAMBS	
$\theta = \{\theta_1, \dots \theta_i\}$	Set of incorporated CLAMBS Managers.
$\forall = \{\forall_1, \dots \forall_i\}$	Set of incorporated CLAMBS Agents.
δ	Number of communication messages between CLAMS components.

Π	Maximum Network Bandwidth for a data-center.
M	Size of CLAMBS message.
π	CLAMBS Agents located in same datacenter of CLAMBS Manager.

Table 5-2: Model analysis notation.

Furthermore, Figure 5.9, presents the class diagram for the proposed framework CLAMBS deployment.

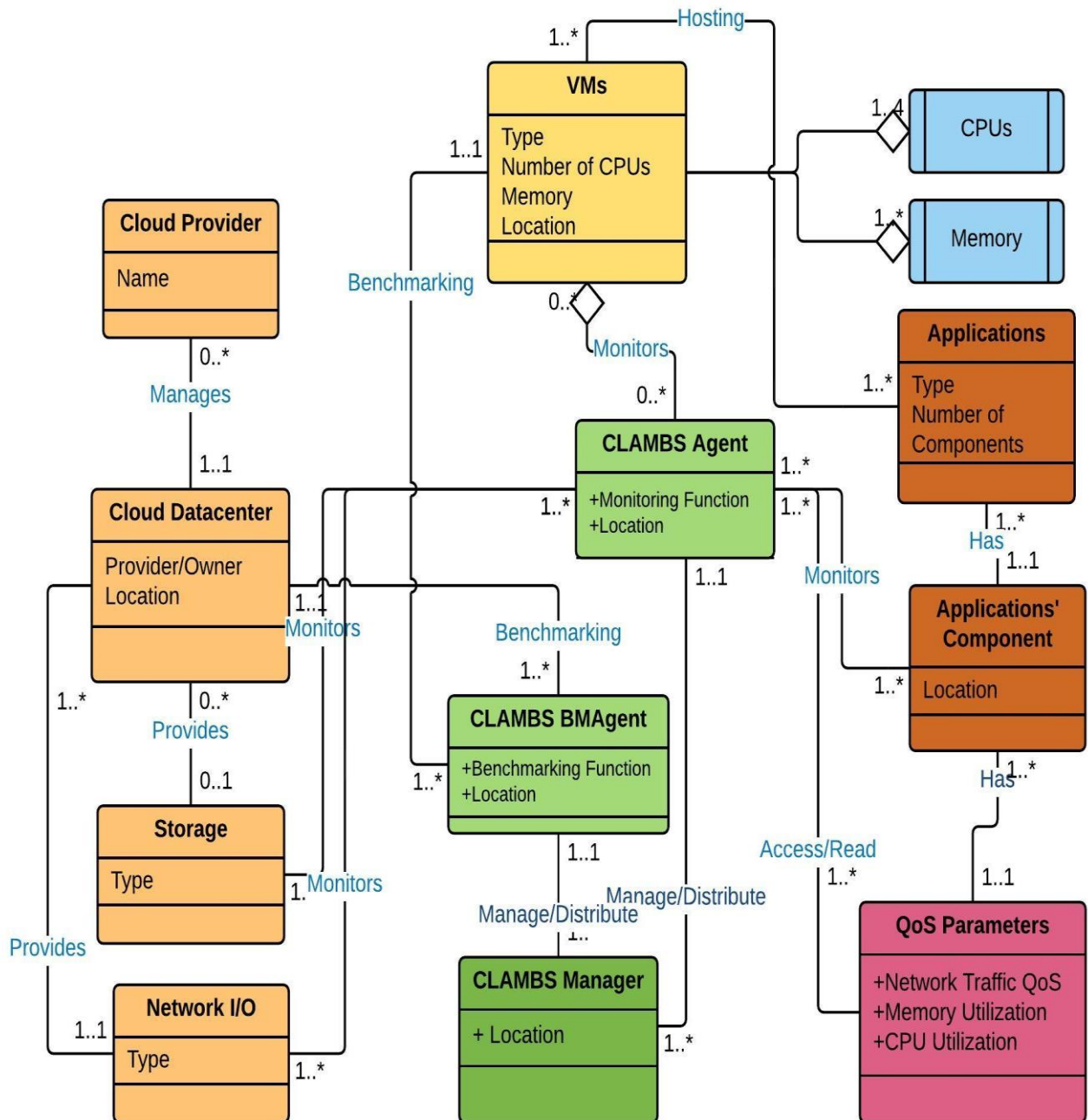


Figure 5.9: CLAMBS framework deployment class diagram.

5.3.2. Communication Overhead

The communication overhead depends on the physical locations of CLAMBS Managers, i.e. datacenter where CLAMBS Agents are distributed across n datacenters. D_1, D_2, \dots, D_n . For a datacenter D_i , there are m_i VMs running: $V_{i,1}, \dots, V_{i,m_i}$. As each VM is accompanied by a CLAMBS Agent, the Agents are denoted as $\forall_i = \forall_{i,1}, \dots, \forall_{i,m_i}$. The size of one CLAMBS Agent message from $\forall_{i,j}$ is $M_{i,j}$. The location and deployment of CLAMBS Agents and Managers will vary. When there is one CLAMBS Manager θ located on datacenter $D_\pi, \pi \in [1, n]$ (See Figure 5.10), each of the distributed CLAMBS Agent $\forall_{i,j}$ on VMs has to communicate with the CLAMBS Manager independently based on a pre-determined time frequency. Thus, the total communication overhead from CLAMBS Agents to CLAMBS Manager in one report will be the total number of messages produced by all CLAMBS Agents except those CLAMBS Agents running on the same datacenter where the CLAMBS Manager is hosted as following:

$$\sum_{j=1}^m ((\sum_{i=1}^n M_{i,j}) - M_{\pi,j}), i = 1, \dots, n; j = 1, \dots, m \quad (1)$$

Practically, the size of CLAMBS communication messages varies between 80-90 bytes. If the message size is a fixed value M , then CLAMBS messages communication overhead is:

$$M \cdot ((\sum_i m_i) - m\pi), i = 1, \dots, n \quad (2)$$

In the above formulas, the messages of Agents located in (π) are excluded being in the same datacenter where CLAMBS Manager is running. Furthermore, for optimization, these messages may not be needed for every report. This scenario will take place if CLAMBS Agent applies data analysis on the messages before sending them to decide if they need to be sent or neglected. Therefore, when changes occur to data, they will be reported to CLAMBS Manager. Thus, if only a subset S_i of V_i is reporting each time, CLAMBS communication cost will be reduced greatly.

If Π_i is the bandwidth (connection speed) for datacenter D_i , the total time consumption in communication (when all CLAMBS messages are sent simultaneously at fixed time slots) is:

$$\text{MAX}_i \left(\text{MAX}_j \left(M_{i,j} \frac{1}{\Pi_i} \right) \right) \quad (3)$$

Therefore, it is possible to develop adaptive algorithms to reduce reports from Agents $\forall_{i,i}$ with large $\Pi_i \cdot M_{i,i}$ to save time, at the cost of CLAMBS messages information. As they are all variable, the criteria could be an average from history. This is a possible way to decide S_i for every Agent report.

When there are n distributed CLAMBS Managers/Sub-Managers located across different data centers (See Figure 5.11), the cost is significantly reduced. Ideally, n managers $\theta_1, \theta_2, \dots, \theta_n$ are located in different data centers. Although management task is distributed, a Super-Manager is still needed for maintaining a centralized database. For example, if the Super-Manager is $\theta_\pi \in \{\theta_1, \theta_2, \dots, \theta_n\}$, in this case, if the message size from θ_i is M_i , then the total communication overhead for each round is reduced to $\sum_i M_i$. However, the optimization in communication overhead also brings other trade-offs or compromises such as in setting up and switching additional Managers, CPU load, response time, etc. I now discuss this further in the following section.

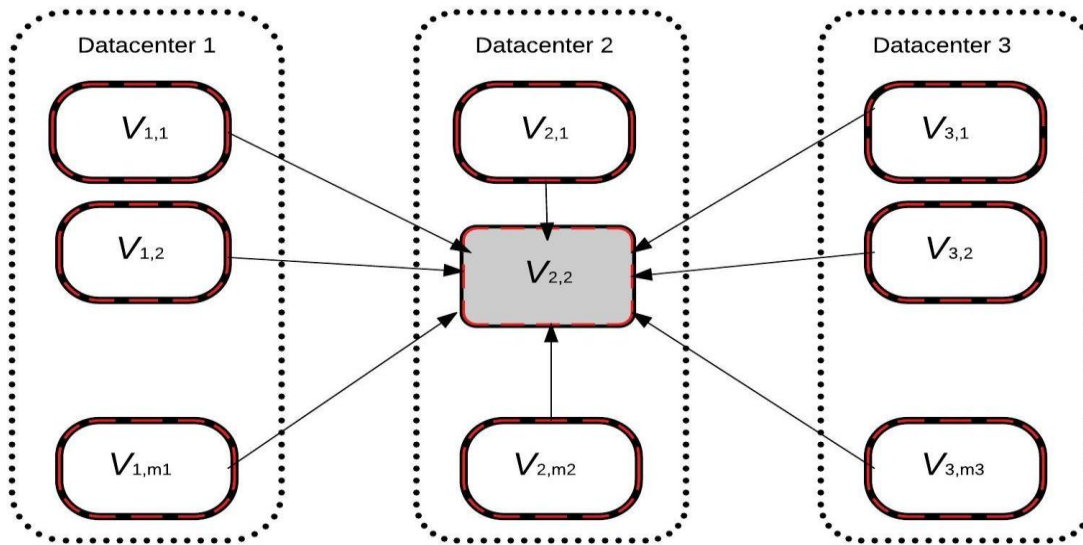


Figure 5.10: Communications: 3 data centers, Manager θ located on $V_{2,2}$.

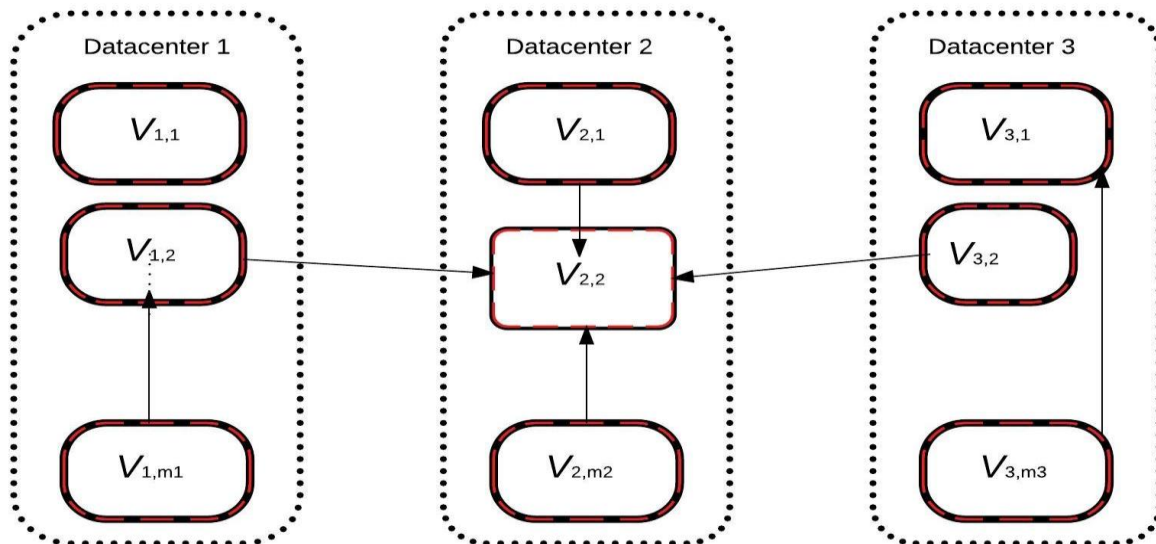


Figure 5.11: Communications: 3 data centers, Managers θ located on $V_{1,2}$, $V_{2,2}$, and $V_{3,1}$.

For further demonstration of the messages communication overheads in CLAMBS framework, two scenarios studied. The first scenario is the centralized deployment where CLAMBS deploys only one Super-Manager. The second scenario is decentralized deployment where CLAMBS has multiple Super-Managers.

Suppose there are a total of 90 nodes presenting the CLAMBS framework deployment across 3 different datacenters as shown in table 5.3. In the first scenario there will be one Super-Manager and 89 CLAMBS Agents. In second scenario, there will be three Super-Managers and 87 CLAMBS Agents.

Scenario	Deployment layout	Datacenter 1	Datacenter 2	Datacenter 3	Number of Super-Managers	Location of Super-Managers
Scenario 1	Centralized	30	30	30	1	Datacenter 1
Scenario 2	Decentralized	30	30	30	3	Datacenter 1, 2, and 3

Table 5-3: Two scenarios for CLAMBS deployment layout.

Supposing that message size is fixed as 100 KB, the frequency is 1 second, then according to equations 1 and 2, the messages communication overhead will be as presented in table 5.4.

Scenario	Communication Overheads
Scenario 1	6000 KB/Sec
Scenario 2	200 KB/Sec

Table 5-4: Messages communications overheads.

5.3.3. CPU load, Response and Search Time

The distributed CPU load will be determined by the layout of CLAMBS Agents. This section will also compare the standard one-Manager layout (model (1), see Figure 5.12) against the hierarchical tree-typed manager structure (model (2 & 3), see Figures 5.13, and 5.14). The total number of CLAMBS Agents is N and the max number of child nodes per node is n .

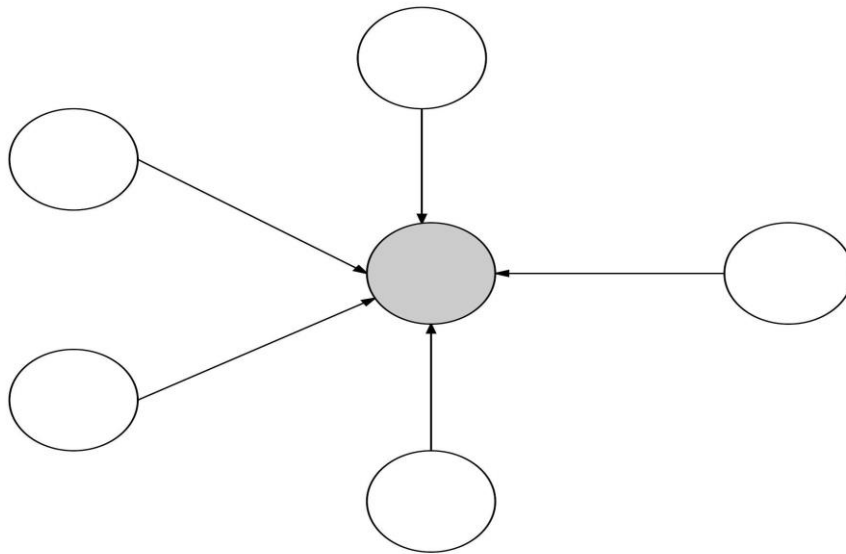


Figure 5.12: Different management structures for 17 CLAMBS Agents – Model 1.

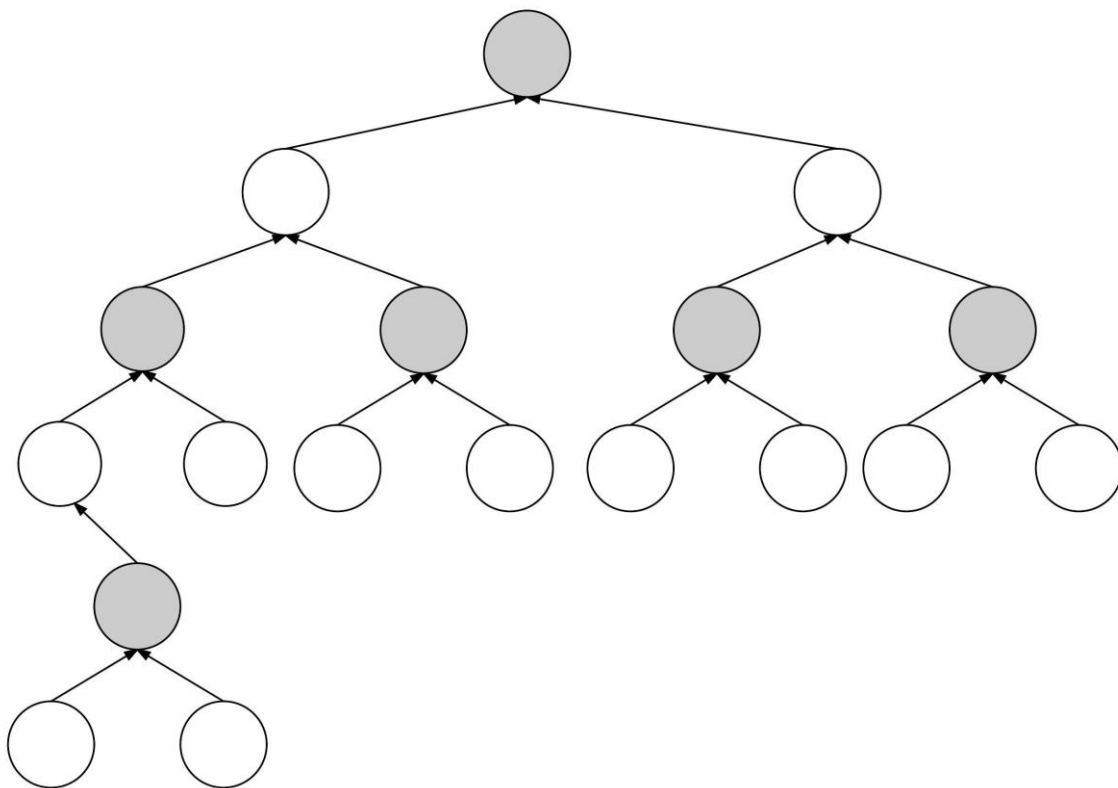


Figure 5.13: Different management structures for 17 CLAMBS Agents – Model 2.

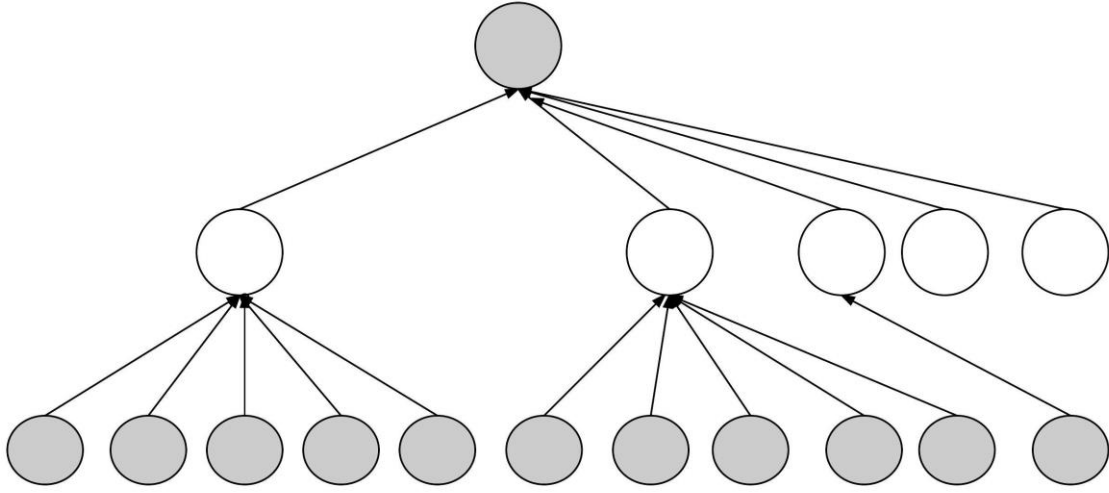


Figure 5.14: Different management structures for 17 CLAMBS Agents – Model 3.

The CPU load for managing one CLAMBS message is C . If there are a total of l levels of the tree control structure, then:

$$l \geq \lceil \log_n (N \cdot (n - 1) + 1) \rceil \quad (4)$$

the inequality turns into an equality when the tree is a complete tree in its top $l - 1$ levels. In model (1) (Figure 5.12), the CPU load for the Super-Manager per round is $(N - 1) \cdot C$ and other nodes is 0. In model (2) (Figure 5.13), the maximum CPU load for Super-Manager will be C , and at least $\lceil (N - 1)/n \rceil$ other Managers will also take over a maximum CPU load of C each. Whatever the load distribution, as the same total number of Agents are returning the same amount of CLAMBS data, the overall CPU load will remain the same. In other words, a larger n will incur less Managers to par-

ticipate and increase the load for each Manager. A smaller n will improve the distribution, but l will also increase so that the response time will grow.

The response time will be determined by the time for a node used to reach the Super-Manager for it to react on unusual behaviors. If the time for a node (Agent) \forall to contact its Manager is t (including processing and communication), then in (1) all response time is t . In (2), the response time will grow for most nodes. The response time for node \forall will be $l_{\forall} \cdot t$ where l_{\forall} is the level of \forall . Under this model, it is easy to observe that a larger n will cause less number of higher-response-time nodes, therefore smaller total response time. As the response time for most individual nodes will grow, the total response time for all N nodes will also grow. Instead of $(N - 1)t$, the total time t_{total} satisfies:

$$\begin{aligned}
t_{\text{total}} &\geq t \cdot \left(\sum_{i=1}^{l-2} i \cdot n^i + (l-1) \cdot \left(N - \sum_{j=0}^{l-2} n^j \right) \right) \\
&= t \cdot \left(\frac{(l-2)n^l - (l-1)n^{l-1} + n}{(n-1)^2} + (l-1) \right. \\
&\quad \left. \cdot \left(N - \frac{n^{l-1} - 1}{n-1} \right) \right) \\
&= t \cdot \left((l-1) \cdot N - \frac{n^l - ln + l - 1}{(n-1)^2} \right)
\end{aligned} \tag{5}$$

Therefore, the average response time t_{ave} for $N-1$ nodes other than the Super-Manager satisfies:

$$t_{\text{avg}} \geq \frac{t}{N-1} \cdot \left((l-1) \cdot N - \frac{n^l - ln + l - 1}{(n-1)^2} \right) \tag{6}$$

As before, the inequalities turn into equalities if and only if the tree is a complete tree in the top $l-1$ levels. In case of a fixed N , when n decreases or l increases, the average response time will grow. Note that here, t is considered a constant value. In practice, communication overhead will also affect the response time of each node. Therefore, minimizing inter-data center communications as shown in communication overhead analysis will also help in lowering response time.

Another metric is the average search time. Similar to a search tree, the (minimum) average search time for the Super-Manager to find a leaf node in (2) is $\log_n N$ (for a complete tree), as opposed to $((N-1)/2) \cdot t$ in (1). Therefore, the search time will also benefit from a larger n .

To sum up, in this section we presented an evaluation of the possible overheads of the CLAMBS framework. The overheads included communications, CPU load, Response, and search time in two different scenarios. The aforementioned scenarios presented the possible methods can be adopted to deploy CLAMBS. This will enable the decision maker to choose how to deploy CLAMBS. The two deployments of CLAMBS Agents have their own advantages and disadvantages. To achieve deserved performance, the system setup will depend on the actual requests and different metrics such as communication overhead, CPU load distribution, average response time analyzed in this section.

5.4. Summary

This chapter has presented implementation details of the CLAMBS framework proposed in chapters 3 and 4, namely, CLAMBS: Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework.

A prototype of the CLAMBS framework has been implemented on real-world cloud platforms, namely, Amazon EC2, and Microsoft Azure platforms. The development programming language was mainly Java, utilizing its rich and supported open-source packages: RESTful, SNMP, and SIGAR. Moreover, the chapter presented modeling and analysis overheads of the prototype in Multi-Cloud Environments. Also, the CLAMBS framework proved the real-world feasibility of cloud applications' components monitoring and benchmarking.

6. Experimentation and Evaluation

6.1. Introduction

Chapter 5, section 5.3 presented a modeling approach for analyzing CLAMBS Overheads in Multi-Cloud Environments. The analysis helps to choose a deployment model for CLAMBS on a cloud environment. In this chapter, to evaluate the CLAMBS framework, experiments were conducted on real-world platforms, namely, Amazon AWS and Microsoft Azure platforms.

To validate CLAMBS Monitoring Agents against overheads while monitoring (see research question 2, section 1.3), we applied four different scenarios. The first scenario is to test CLAMBS overheads incorporating 5 CLAMBS Agents. The second scenario is to Test CLAMBS overheads incorporating 30 CLAMBS Agents. The third scenario is to Test CLAMBS overheads incorporating 50 CLAMBS Agents. The fourth scenario is to Test CLAMBS overheads incorporating 85 CLAMBS Agents. These scenarios incorporate different number of running Agents in order to detect the overheads while communications in each scenario.

To validate CLAMBS Benchmarking feasibility during benchmarking network performance between different locations (see research question 3, section 1.3), we did different benchmarks. First we applied data download latency benchmark. Second, we applied upload latency benchmark. Third, we applied Download/Upload bandwidth benchmark. The outcomes of these different benchmarks help to validate how feasible is CLAMBS to conduct network benchmarking between distinct locations. The outcomes were different as expected depending on the locations of different networks.

To validate CLAMBS Manager Scalability under Benchmarking (see research question 2, section 1.3), we measured the CLAMBS Manager performance. This is, we measured the CLAMBS Manager CPU and Memory utilization while benchmarking. By doing this, we could validate how CLAMBS is scalable and robust while performing its tasks.

The summary of the aforementioned objectives are presented in Table 6.1 below.

Objectives	Scenario	Section
Validate CLAMBS Monitoring Agents against overheads while monitoring. (see research question 2, section 1.3)	Test CLAMBS overheads incorporating 5 CLAMBS Agents.	6.4.1
	Test CLAMBS overheads incorporating 30 CLAMBS Agents.	
	Test CLAMBS overheads incorporating 50 CLAMBS Agents.	
	Test CLAMBS overheads incorporating 85 CLAMBS Agents.	
Validate CLAMBS Benchmarking feasibility during benchmarking network performance between different locations. (see research question 3, section 1.3)	Data download latency benchmark.	6.4.2
	Data upload latency benchmark.	
	Download/Upload bandwidth benchmark.	
CLAMBS Manager Scalability under Benchmarking. (see research question 2, section 1.3)	Measuring CLAMBS Manager CPU and Memory utilization while benchmarking.	6.5

Table 6-1: Experiments objectives and evaluation.

6.2. Hardware and Software Configuration

Standard small instances were used on each platform. The AWS instance has the following configurations: 619 MB main memory, 1 EC compute unit, 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 configuration: 768 MB main memory, 1GHz CPU (Shared virtual core) and a 64 bit platform. Three different datacenters are considered in this experiment, namely, Sydney, US-Virginia, and Singapore. The CLAMBS Manager was located in Sydney. One CLAMBS Agent was hosted on a VM at US-

Virginia datacenter and another CLAMBS Agent was hosted on a VM in Singapore datacenter. Figure 6.1 presents a view for the distributed CLAMBS components across the aforementioned datacenters.

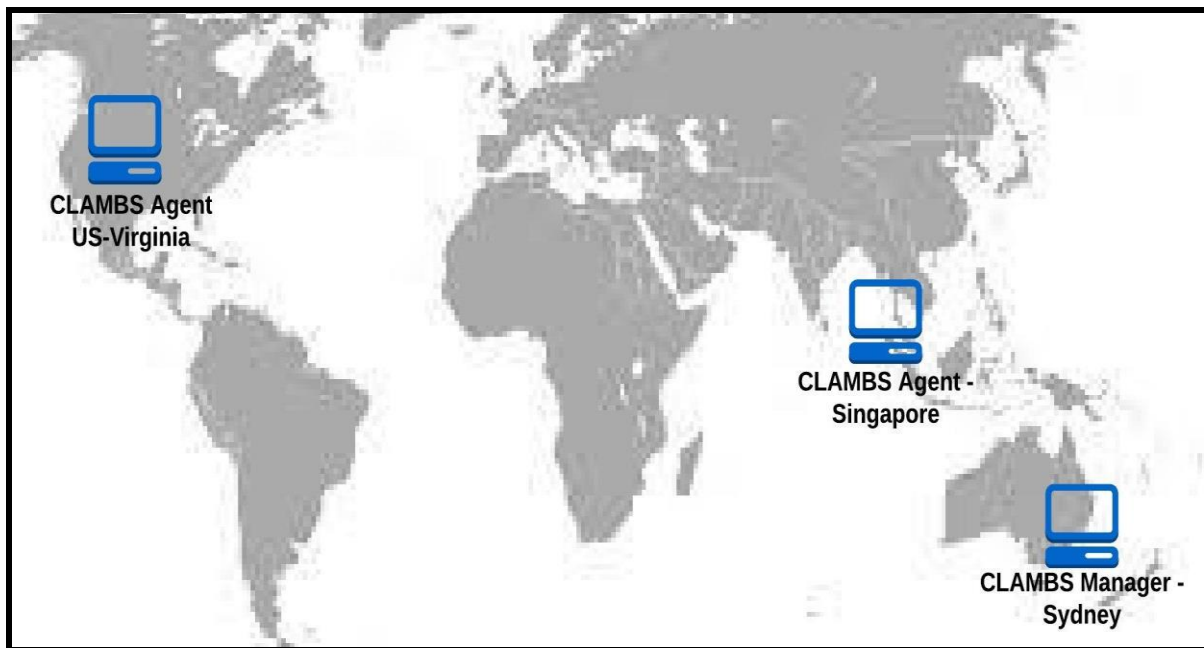


Figure 6.1: Distributed CLAMBS components across datacenters.

All VMs in the experiments were running Microsoft Windows Operating System. For persistent storage of CLAMBS Agent and Manager data (Figure 3.2), off storage volumes such as Elastic Block Store (EBS) in Amazon EC2 and XDrive in Windows Azure were used. 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; and ii) storage volumes

offer the ability to create point in time snapshots, which could be extended to the cloud specific data repositories.

6.3. Experimental Setup

As discussed previously (see section 5.1), the CLAMBS framework has three main components namely the CLAMBS Manager, CLAMBS Monitoring Agent and CLAMBS Benchmarking Agent. This section presents the experimental scenario and setup of the Monitoring and Benchmarking Agents. In both cases, the Manager is responsible for collecting monitored and benchmarked QoS parameters.

To evaluate and validate CLAMBS prototype, the scenario of a web audio/video streaming application that uses a content distribution network to distribute multimedia content to end-users using a multi-cloud provider setup (e.g. combination of Amazon AWS and Windows Azure) was considered. The CLAMBS prototype was used to benchmark and monitor the performance of this audio/video streaming application components, namely the search and indexing server (Tomcat web server and MySQL database) and network QoS parameters including network latency and download and upload performance.

6.3.1. CLAMBS Monitoring Agent Setup

Each Monitoring Agent comprises the corresponding SNMP and SIGAR package dependencies to accomplish the monitoring task (see sections 5.2.1.4 and 5.2.1.5). In the experiment, the Monitoring Manager triggered a request to Monitoring Agents, which in turn retrieved the requested QoS parameters from the hosted VM. Each Monitoring Agent running on the VM listened 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 CLAMBS cross-layers monitoring capability, each Monitoring Agent monitored several resources including system resources and user processes. Table 6.2 presents the list of monitored processes/resources. On retrieving QoS data from the Monitoring Agents, the Monitoring Manager saves the data into a local database by classifying them as system performance or user applications QoS performance parameters.

Process/Resource	Description	Owner
Tomcat7w.exe	Apache Tomcat 7	User
MySqlld.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

Table 6-2: Monitoring various resources across different layers.

6.3.2. CLAMBS Benchmarking Agent Setup

The Benchmarking Agent is composed of two components which are network traffic benchmarking and CLAMBS load generator (see section 5.2.3.2). Similar to Monitoring Agent, each Benchmarking Agent comprises the corresponding required Java package dependencies to accomplish the benchmarking task. In this experiment setup, the network QoS parameters that links between the CLAMBS Manager and the Benchmarking Agents are tested. Benchmarking the network link connecting a Benchmarking Agent and the CLAMBS Manager was accomplished by generating bi-directional traffic to simulate download and upload processes. This experiment

was run to demonstrate CLAMBS ability to benchmark network performance between two different locations of datacenters.

In the experiments, the CLAMBS Manager triggered the benchmarking requests to CLAMBS Benchmarking Agents, which responded immediately to the Manager's request. Communications between CLAMBS Manager and Agents were conducted using the RESTful HTTP protocol. Pre-defined files with varying sizes (50 MB, 100MB, and 200MB) were used during the experiment to measure network performance over a download/upload process. Table 6.3 lists the measurements parameters that were observed throughout the experiment. According to the proposed conceptual framework, such measurements provide the user with the ability to decide and choose a preference for what site/location a service is performing better. Likewise, a service provider will acquire such knowledge in order to improve the delivered service quality to clients.

Traffic Benchmarking Measurement Parameter	Description
Download File Network Latency Time	Time consumed starting from a request up-till download complete including Network Latency
Upload Network Bandwidth	Amount of data transferred per Second while download process
Upload File Network Latency Time	Time consumed starting from a request up-till upload complete including Network Latency
Upload Network Bandwidth	Amount of data transferred per Second while upload process

Table 6-3: Benchmarking parameters measurements parameters.

6.3.3. Runtime Configuration Monitoring Agent

CLAMBS Monitoring Agents (section 5.2.3.2) as well as CLAMBS Manager are packaged into jar files with corresponding dependencies and configured to run during the 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.

Figure 6.2 provides a detailed workflow of communication between the CLAMBS Monitoring Manager and Agents. The Monitoring Manager instantiated parallel threads for each group of Agents in one VM; that is, each thread was dedicated to only one VM to communicate with Agents running on that VM. The Manager thread sent requests to Agents addressed by IP address and port numbers. The request was for a list of QoS parameters Monitored by the Agent. After receiving the request, Agents computed the QoS parameter values from the hosting VM. The Agents then responded to the Manager with corresponding QoS parameters.

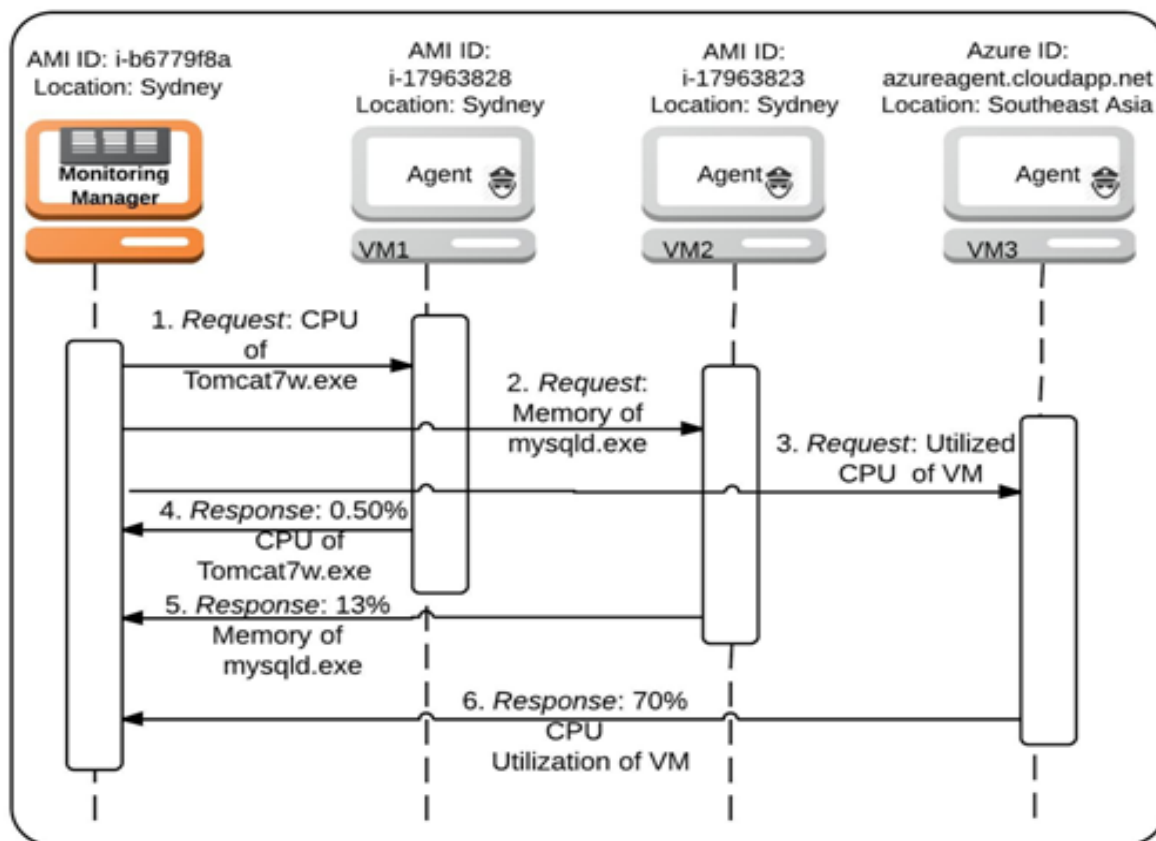


Figure 6.2: CLAMBS Manager/Agents run-time communication workflow.

The CLAMBS Agents and Manager were deployed on four virtual machine instances (3 VM's on AWS platform and 1 on Microsoft Azure platform). On VMs that hosted the Agent, depending on number of Agents, the Agents were bound to unique ports. For example, if VM-3 hosted 30 Agents, it was bound to ports 8001-8030. Similarly if VM-4 hosted 10 Agents, it was bound to ports 8001-8010.

6.3.4. Runtime Configuration Benchmarking Agent

CLAMBS Manager and Benchmarking Agents (section 5.2.3.3) are packaged into runnable jar and war files with corresponding dependencies and configured to run during the VM boot process. The Agents use a configuration file that is required to run and remain on standby waiting for the Manager requests. Intervals of requests can vary, but initially it is set to 10 seconds for each request sent to a single CLAMBS Agent. Agents in turn immediately respond to CLAMBS Manager requests. Fixed data with pre-chosen sizes are stored locally in each VM hosting CLAMBS Manager and CLAMBS Agents to be utilized for data transferred during the experiment. The CLAMBS Manager instantiated parallel threads for each CLAMBS Agent (see section 5.2, figure 5.8) addressed by IP address and port number. Concurrently, CLAMBS Manager sends similar requests to other registered CLAMBS Agents in different datacenters which can also be for a different cloud platform provider.

6.4. Experimental Results and Discussion

6.4.1. CLAMBS Monitoring Agent

To validate that the CLAMBS Monitoring Agent does not introduce significant overheads while monitoring QoS parameters across-layers in multi-cloud environments, experiments were executed on four typical multi-cloud workload scenarios described in table 6.4. In all scenarios, Agents were deployed on multi-cloud environments (3 AWS instances and 1 Azure instance).

Workload Scenario	VM-1	VM-2	VM-3	VM-4
I	hosts the CLAMBS Manager	Hosts 1 Agent	Hosts 1 Agent	Hosts 3 Agent
II	hosts the CLAMBS Manager	Hosts 10 Agents	Hosts 10 Agents	Hosts 10 Agents
III	hosts the CLAMBS Manager	Hosts 10 Agents	Hosts 20 Agents	Hosts 50 Agents
IV	hosts the CLAMBS Manager	Hosts 25 Agents	Hosts 30 Agents	Hosts 30 Agents

Table 6-4: Experimental workload scenarios.

For each scenario, the CPU and memory consumption of the CLAMBS Manager were monitored. Figures 6.3 and 6.4 present the results of the experiments. The av-

erage CPU and memory utilization by the Manager is computed for each scenario. Each evaluation scenario involving communication between Agents and Manager was run for a duration of 30 minutes. The frequency of querying the Agents for QoS parameters was set to 1 second. The outcomes clearly indicate that the CLAMBS Manager performance is stable with an increase in the number of active Agents. The CPU utilization increased from 6.25% when Manager was communicating with 5 Agents to 10.92% when the number of Agents was 85. Likewise, the amount of memory consumed by the CLAMBS Manager increased marginally from 177.5 MB with 5 Agents to 177.85 MB with 85 Agents. Moreover, it was noted that, the CLAMBS Manager or the Agents during the experiment did not encounter any crash or malfunction. These outcomes clearly validate the resource efficient operation of the CLAMBS prototype and its ability and suitability to scale across multi-cloud environments.

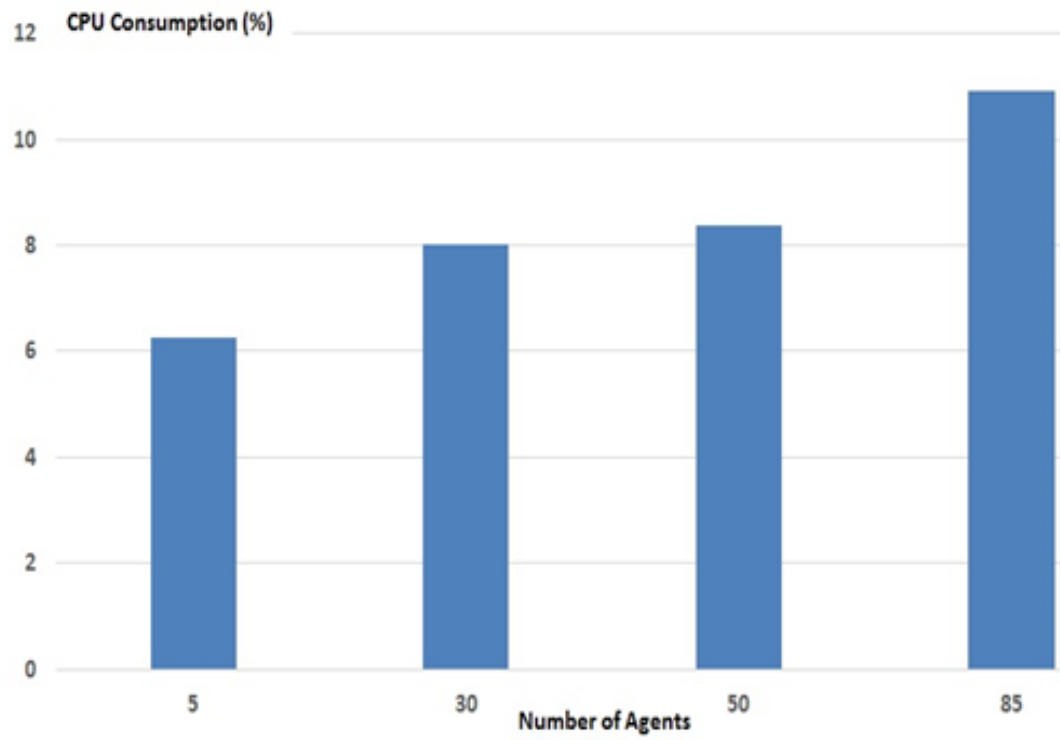


Figure 6.3: Manager CPU consumption in percentage (Monitoring Scenario).

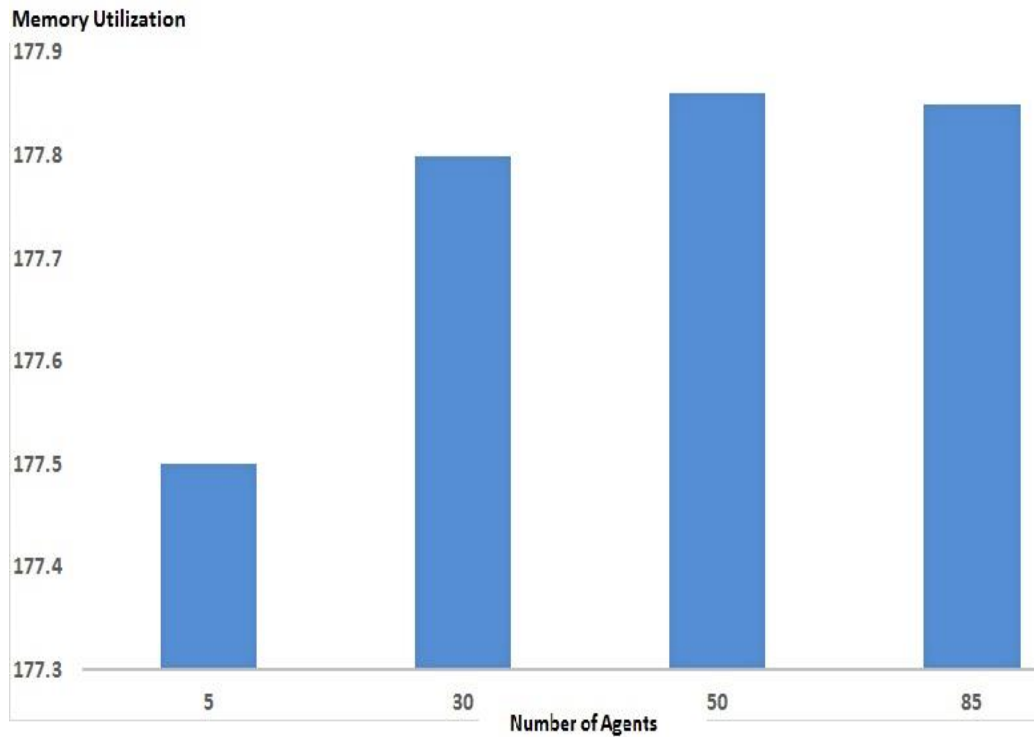


Figure 6.4: CLAMBS Manager Memory utilization in MB.

6.4.2. CLAMBS Benchmarking Agent

To demonstrate CLAMBS benchmarking ability, the network performance between datacenters in different locations is benchmarked based on the experimental setup presented earlier.

6.4.2.1 Data Download Latency Benchmark

Concurrently, CLAMBS Manager started downloading data from Agents in Singapore and US-Virginia datacenters. Each request indicates what size of data is to be downloaded (50MB, 100MB, or 200MB). As presented in figure 6.5, the CLAMBS

Agent in Singapore datacenter provided faster data download compared to CLAMBS Agent in US-Virginia. Moreover, it was observed that as the data size increased, the data transfer latency from CLAMBS Agent in US-Virginia also increased. Such observations are expected to have a major impact on both service provider and service client.

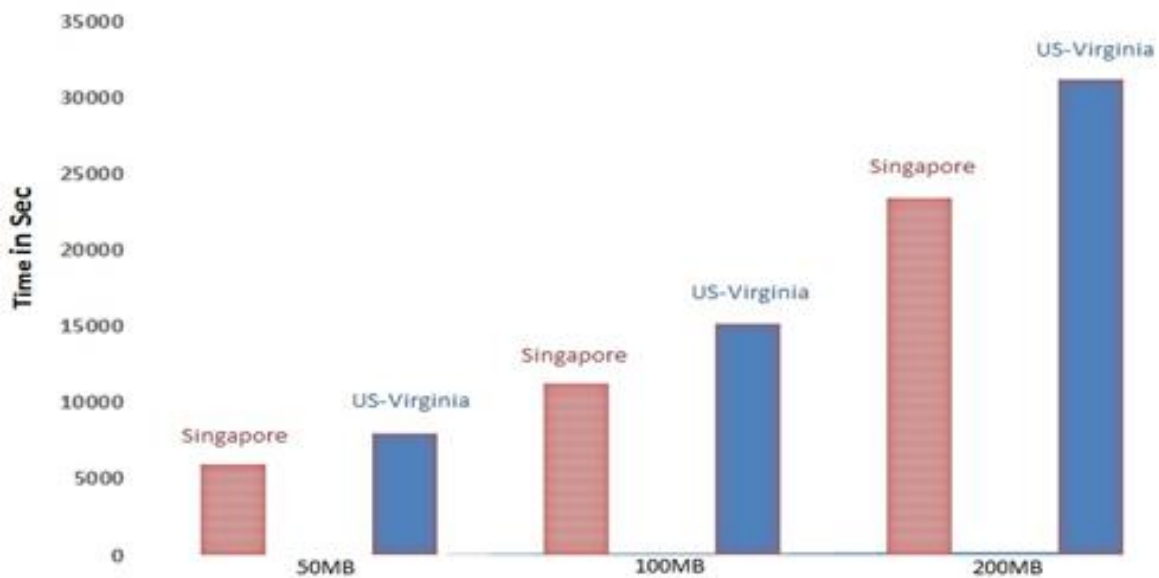


Figure 6.5: Data Download Network Latency (Time in Seconds).

6.4.2.2 Data Upload Latency Benchmark

Experiments, as shown in figure 6.6 demonstrates how network traffic benchmarking has the potential to drive preferences of both service provider and service client. Uploading 50MB, 100MB, and 200MB files from Sydney datacenter to Singapore dat-

acenter show shorter latency times compared to uploading the same size of data to US-Singapore datacenter.

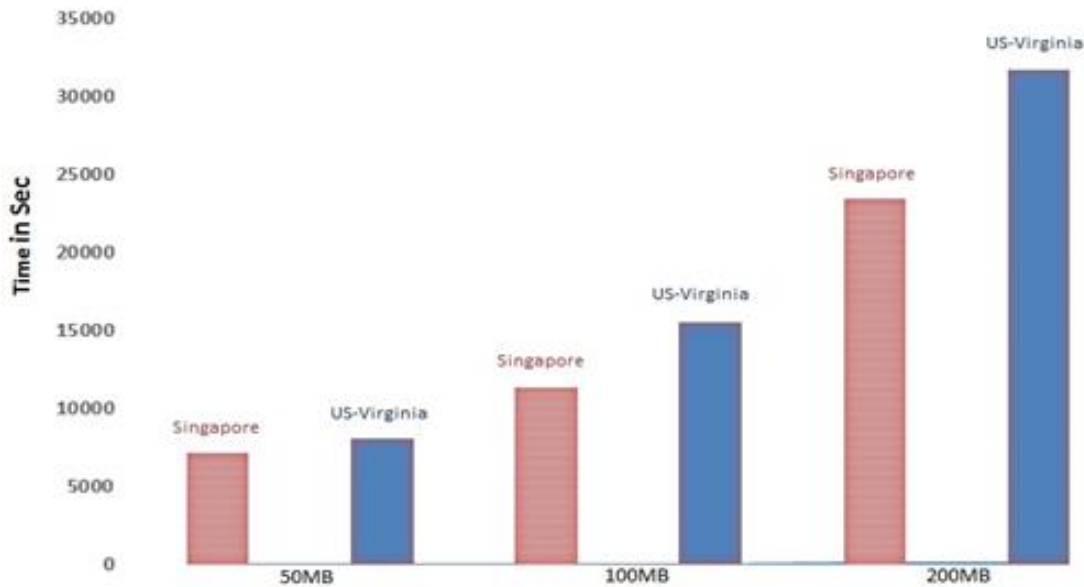


Figure 6.6: Data Upload Network Latency (Time in Seconds).

6.4.2.3 Download/Upload Bandwidth Benchmark

Experimental results as shown in figure 6.7, presents the outcome of upload/download bandwidth between Singapore, Sydney and US-Virginia datacenters. With 50MB, 100MB, and 200MB size of data being transferred, network bandwidth between Sydney and Singapore remains the same at 8 KB/s. Similarly, the network bandwidth between Sydney and US-Virginia is 6 KB/s for the different data sizes transferred. This demonstrates that the CLAMBS benchmarking capability enables the user to prefer one location over another. In this experimentation scenario the

Singapore datacenter site measured a significantly better performance over the US-Virginia datacenter.

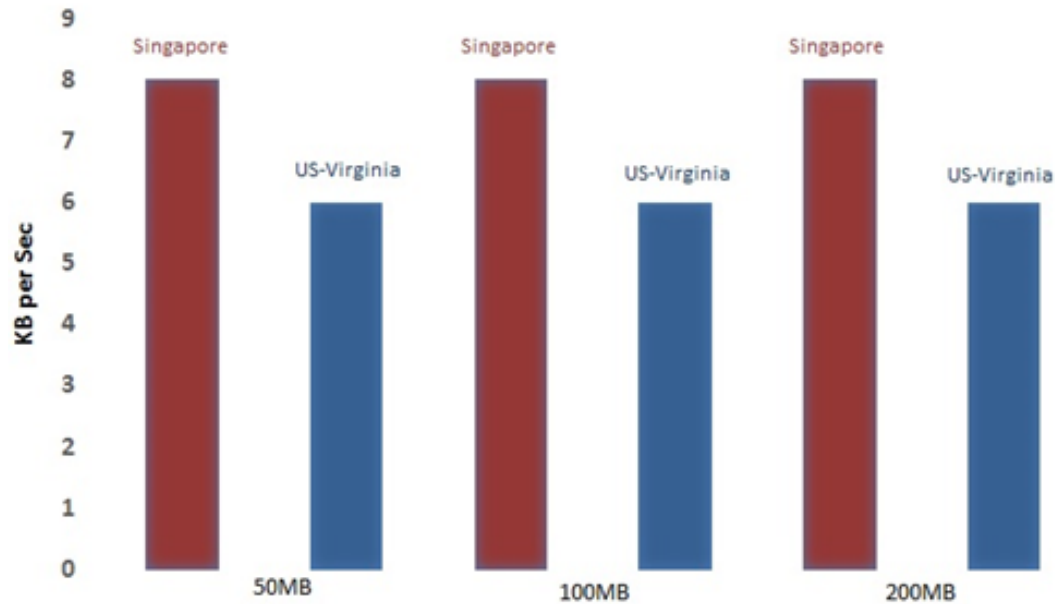


Figure 6.7: Download/Upload Bandwidth (Kilobytes per Seconds).

6.5. Experiments Scenarios Analysis for CLAMBS validation and Feasibility

This section will present analysis for CLAMBS scalability and demonstrates the encountered limitation during the experiments.

6.5.1. Development Environment Limitations

Referring to AWS documentation³², network performance for small instance types are low. Moreover, such types of instances are not listed under eligible instances for enhanced network performance. Unlike other instance types (e.g. c3.large, c3.xlarge, c3.2xlarge, c3.4xlarge, c3.8xlarge, i2.xlarge, i2.2xlarge, i2.4xlarge, i2.8xlarge, r3.large, r3.xlarge, r3.2xlarge, r3.4xlarge, or r3.8xlarge), small instance types do not have a feature of enabling enhanced network performance. This limitation was addressed in the experiments by having low network bandwidth across different datacenters. Furthermore, VM requests serving priority by the hosting server on Amazon platform are low, which means that the performance is minimal for such small instances.

6.5.2. CLAMBS Manager Scalability under Benchmarking

The average CPU and memory utilization by the CLAMBS Manager was computed while performing benchmarking of an application's network performance. A file size of 100 KB enabled the operation of data transfer between CLAMBS Manager and Agents located in different remote datacenters locations to be repeated. In this scenario, a CLAMBS Monitoring Agent to monitor the performance of the CLAMBS Manager was used. As indicated by the experimental outcome, and similar to the

³² <https://aws.amazon.com/documentation/aws-support/>

CLAMBS Manager's performance (see figures 6.3, and 3.4) while monitoring, the overheads imposed by the benchmarking component of the CLAMBS Manager on the underlying system memory consumption are not very significant as shown in figure 6.8 . Moreover, the CPU consumption of CLAMBS Manager during benchmarking scenario was also not significant and ranged between 2 – 5%.

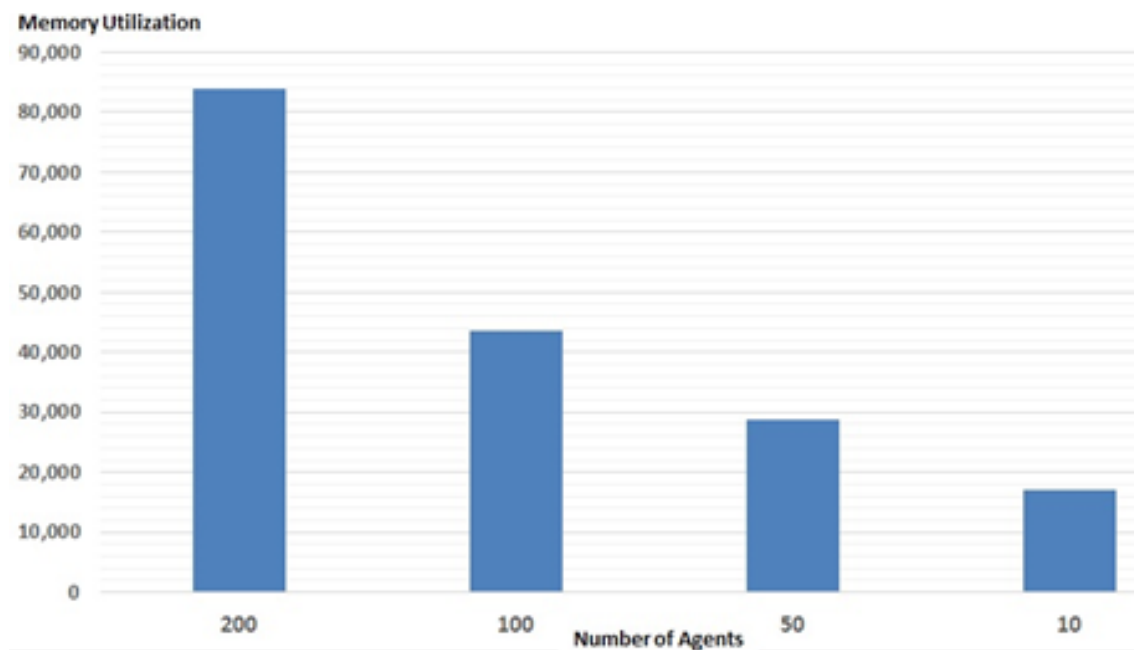


Figure 6.8: CLAMBS Manager memory consumption (benchmarking scenario).

The experimental outcomes validate the CLAMBS framework's ability to be reliable in benchmarking network traffic across multiple datacenters using different

sizes of transferred data. The next section will present a summary of all outcomes of the experiments stated earlier in this chapter.

6.6. Summary

This chapter has presented the outcomes of experiments performed to verify and validate the CLAMBS framework proposed in chapters 3, 4 and 5. Table 6.5 presents the outcomes summary for the stated objectives of the CLAMBS framework during the experiments.

Objectives	Outcomes	Section
Validate CLAMBS Monitoring Agents against overheads while monitoring.	The CLAMBS Manager performance is stable with an 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, the amount of memory consumed by the CLAMBS Manager increased marginally from 177.5 MB with 5 Agents to 177.85 MB with 85 Agents.	6.4.1
Validate CLAMBS Benchmarking feasibility.	CLAMBS Agent in Singapore datacenter provided faster data download compared to CLAMBS Agent in US-Virginia.	6.4.2
	Uploading data from Sydney datacenter to Singapore datacenter show shorter latency times comparing to uploading the same size of data to US-Singapore datacenter.	
	Network bandwidth between Sydney and Singapore remains the same at 8	

	KB/s. Similarly, the network bandwidth between Sydney and US-Virginia is 6 KB/s for the different data sizes transferred.	
CLAMBS Manager Scalability under Benchmarking.	The overheads imposed by the benchmarking component of the CLAMBS Manager on the underlying system memory consumption is not very significant.	6.5

Table 6-5: The experimental outcomes summary.

The experimental evaluations of the proposed CLAMBS framework leveraging different development technologies, tools and in real-world environment have significant potential for monitoring and benchmarking cross-layers applications' components on multi-cloud environments. Experimentation and the prototype implementation show that CLAMBS is flexible, scalable and resource efficient and can be used to monitor and benchmark several applications and cloud resources distributed across multiple clouds.

7. Conclusion and Future Work

Cloud applications monitoring and benchmarking is a key research area that raises a number of unique technical challenges. Cloud applications monitoring and benchmarking plays a vital role in developing provisioning techniques for guaranteeing SLAs. Cloud applications' components are distributed across multiple-layers and on multi-clouds. Hence, this thesis proposed, designed, formulated, and developed a unique and novel monitoring and benchmarking framework and techniques which provide the required awareness of performance SLA QoS targets for cloud hosted applications.

This thesis has successfully addressed the challenges of cross-layered application monitoring and benchmarking in multi-cloud environments. The thesis ends by highlighting the major contributions and future research directions that can build upon outcomes of this research.

7.1. Contributions of the Thesis Work

7.1.1. Research questions

In chapter 1, section 3.1, three research questions were formulated to address the aim of this thesis:

1. What is the current state-of-the-art architecture dimensions and issues of cloud applications' monitoring and benchmarking? In particular:
 - What is the body of the knowledge in current cloud monitoring and benchmarking tools and techniques?
 - What is the support for multi-cloud and cross layers monitoring and benchmarking?
2. How to design a monitoring tool which is scalable, dynamic, agnostic to cloud platform, agnostic to cloud layer, and agnostic to cloud application type? In particular:
 - How to determine layer specific application monitoring requirements; i.e., how cloud consumers can stipulate at which cloud layer (SaaS or PaaS or IaaS) his/her application should be monitored?
 - How cloud consumers can stipulate on which cloud provider platform or datacentre his/her application should be monitored?
 - How to model QoS and SLA information to monitor applications' performance?
3. How to design a benchmarking tool which closely integrates with a monitoring tool and is able to perform real-time benchmarking of applications' components at SaaS, PaaS, and IaaS layers?

7.1.2. Addressing First Research Question

In relation to the first research question, the thesis surveyed and investigated the body of knowledge in context of cloud application monitoring and benchmarking tools and techniques. Furthermore, it analyzed how current approaches support

cross-layers multi-clouds monitoring and benchmarking. Based on the investigation of the available literature, a taxonomy for classifying current monitoring and benchmarking approaches was developed. Further, some research dimensions that aid in understanding the technical capabilities and limitation of the current generation of monitoring and benchmarking frameworks and techniques were proposed.

7.1.3. Addressing Second Research Question

To address the core technical challenges involved with developing techniques and frameworks that can monitor cloud applications in multi-cloud environments, the thesis in chapter 3 developed and designed a monitoring framework which is scalable, dynamic, agnostic to cloud platforms, agnostic to cloud layer, and agnostic to cloud application type. In particular, the core contributions in this part of the thesis included that:

- The CLAMBS framework, which is able to determine layer specific application monitoring requirements i.e., SaaS, PaaS, and IaaS specific QoS parameters to be monitored.
- The CLAMBS framework enables the user to stipulate how an application should be monitored on a specified cloud provider platform or datacentre.

A proof-of-concept implementation (chapter 5) was presented to validate the feasibility of the proposed framework in real-world scenarios based on the experiments and prototype implementation. In particular, the novel features of CLAMBS include:

- Ability to monitor and profile QoS of applications, whose hardware and software components are distributed across multiple public or private clouds; and
- Ability to provide visibility into QoS of individual components of given application stack (e.g., web server and database server in context of multi-tiered web applications).

7.1.4. Addressing Third Research Question

The CLAMBS framework closely integrates with monitoring tools and is able to perform real-time benchmarking of applications' components at SaaS, PaaS, and IaaS layers in multi-cloud environments.

As demonstrated in chapters three and four, the CLAMBS framework was intended to be cross-layers multi-cloud monitoring and benchmarking framework. Thus, it is worth noting that CLAMBS framework has the following novel features:

- It provides visibility into QoS of individual components of application stack (e.g., CPU at IaaS layer, Database server at PaaS layer, and web application at SaaS layer). In particular, CLAMBS facilitates efficient collection and sharing of QoS information across SaaS, PaaS, and IaaS layers by deploying a cloud provider agnostic intelligent multi-agent technique;

- It provides benchmarking-as-a-service that enables the establishment of baseline performance of application deployed across multiple layers using a cloud-provider agnostic technique; and
- It is a comprehensive framework allowing continuous (real-time) benchmarking and monitoring of multi-cloud hosted multi-layered applications.

Further, to verify, validate and evaluate the proposed CLAMBS framework, in chapters 4 and 5, I;

1. Implemented the Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service (CLAMBS) Framework in Java, SNMP, RESTlet technology, and SIGAR.
2. Demonstrated the scalability and efficiency of CLAMBS by conducting extensive real-world experimentations on cloud platforms such as Amazon AWS, and Microsoft Azure platforms.
3. Presented an empirical evaluation of CLAMBS framework.

As a result of the aforementioned features and capabilities of the CLAMBS framework, system administrators and applications developers have the following capabilities:

- I. keeping the cloud services and applications operating at peak efficiency;

- II. detecting variations in service and application performance;
- III. accounting the SLA violations of certain QoS parameters; and
- IV. tracking the leave and join operations of services due to failures and other dynamic configuration changes.

It should however be noted that the developed framework is quite generic, as it is agnostic to cloud platforms, applications, service and cloud layers.

In addition, an evaluation study was presented to evaluate the CLAMBS performance (section 6.1). Based on different scenarios presented in (table 6.5) and deployments methods, we could evaluate how CLAMBS can perform. The performance was measured through different factors like the communication, CPU load, and Memory utilization. The outcomes provided promising results that validate how CLAMBS is scalable and robust.

7.2. Limitations

Based on the discussed facts and on the aforementioned monitoring and benchmarking aspects and approaches, I believe that considerable effort is required to have more reliable cloud monitoring and benchmarking approaches. Because I found that there is a lack of reachable standards on procedure, format, and metrics to assess the development of cloud monitoring and benchmarking. Mainly, commercial monitoring and benchmarking tools do not provide such technical information for publish access, which makes it challenging to formulate and advancing a new monitoring

and benchmarking tool. For instance, although rich documentation for Cloudwatch monitoring tool is provided by Amazon, the technical information that explain how Cloudwatch performs are hidden. Likewise, Microsoft Azure do not reveal any technical data for the Azure FC.

For the purpose of conducting the experiments, I needed to have access to several VMs provided by at least two major cloud providers. Amazon and Microsoft cloud platforms provide various type user accounts to access the cloud platform resources. Nevertheless, the affordable type of accounts I used provide limited resources accordingly and hence, have the following limitations:

- Limited network resources, this means the VMs will have the very lowest network bandwidth that could be provided. This impacts the communication performance between the distributed VMs among different datacenters during the experiment.
- Limited VM RAM capacity, which presents slowness in the performance of the running applications and the CLAMBS components on a VM. For example, slowness in the CLAMBS database and web server components was notable.
- Limited CPU features and capabilities, that impacts the overall performance of the VM and the required applications running on that VM.

7.3. Future Work

Based on the aforementioned limitations, I recommend having more collaborative use of research facilities in which tools, lessons learned and best practices (and moni-

tored data logs) can be shared among all interested researches and professions. Moreover, the research questions addressed in this thesis have created new opportunities for further research. I highlight some of them in this section.

7.3.1. CLAMBS: Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service Framework

CLAMBS framework can be improved by incorporating additional development technologies. Furthermore, CLAMBS can be integrated within a cloud orchestration framework to provide QoS-awareness for cloud admission control and scheduling of Big Data applications in a highly distributed multi-cloud environment.

Data mining and application programming frameworks provide the ability to formulate a big data analytics application architecture. Largescale data mining frameworks (e.g. GraphLab [103] FlexGP [48] Apache Mahout³³, and MLBase [85], apply many data mining algorithms such as clustering, decision trees, regression, and Bayesian for mining datasets simultaneously by leveraging distributed sets of machines. However, such complicated, dynamic configurable large scale frameworks require novel monitoring and QoS control techniques. Ensuring QoS for such frameworks across cloud layers on multi-cloud environments is a challenging task. QoS parameters are diverse for each computing platform hosting a large scale framework. Key quality factors include throughput and latency in a distributed messaging system, response time in the batch processing platform, and precision recall in the scalable data mining platform. Consequently, we need to know:

³³ <http://mahout.apache.org>

- how these QoS could be defined consistently across layers;
- how the various measures should be combined to give a holistic view of the stream of data flows end-to-end; or
- how optimal optimization would be realized in cases with large sets of variables and constraints, such as with heterogeneous resources, non-steady workloads, and so on.

To this end, future research efforts must take an end-to-end QoS view of large scale frameworks and develop techniques that address all components rather than handling them as one black-box.

7.3.2. Monitoring big data security and privacy

In cloud environments, while implementing the security controls framework, the cloud platform provider can only conduct and process the type of data a customer will actually generate and use. Consequently, the cloud service provider is not aware of the additional security and privacy requirements or custom security controls that are considered to be necessary to protect the customer's data. Equally, customers can obtain only a rough view of the cloud service provider's security policies and the implemented mechanisms. Such limitations are challenging for deploying innovative features such as monitoring and end-to-end security assurance in multi-cloud platforms.

NIST and the European Commission (EC) consider SLAs as the most important element for cloud service providers to establish their sincerity and attract cloud customers because SLAs will be used as a mechanism for service variation. They sug-

gest the use of cloud SLAs to develop better assessments and inform customer decisions, and eventually to advance trust and transparency among cloud users. To empower the SLAs from security and privacy perspectives, multiple users in the cloud community such as the European Network and Information Security Agency (ENISA) [124], the International Standards Organization/International Electro-technical Commission (ISO/IEC) [1], NIST, and the EC have identified that determining security parameters in SLAs, or secSLA, is useful for establishing common semantics to provide and manage security assurance for both cloud service providers and cloud customers.

Based on agreed secSLA between the cloud service provider and the cloud service customer, the cloud customer will have a mechanism to monitor the QoS parameters defined in this secSLA. This mechanism helps to assess the fulfillment of agreed security and privacy objectives or any potential violations.

To the best of my knowledge, few efforts have been made to explore this area. CLAMBS framework with its current cross-layers and multi-cloud monitoring and benchmarking capabilities can be further extended to assess the feasibility of the aforementioned security monitoring approach. Defined secSLA parameters can be QoS targets for CLAMBS to monitor for assuring secSLA objectives and avoiding violations.

References

- [1] "Cloud Service Level Agreement Standardization Guidelines," *EC Cloud Select Industry Group (CSIG), European Commission*, 2014.
- [2] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, pp. 2093-2115, 2013.
- [3] O. Adinolfi, R. Cristaldi, L. Coppolino, and L. Romano, "QoS-MONaaS: A Portable Architecture for QoS Monitoring in the Cloud," in *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, 2012, pp. 527-532.
- [4] 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.
- [5] A. Alexandrescu and P. Marginean, "Generic: Change the Way You Write Exception-Safe Code Forever," *Dr. Dobb's Journal, CMP Media LLC*, 2003.
- [6] Amazon, "Amazon Web Services," <http://aws.amazon.com/>, 2013.
- [7] Amazon, "AWS," http://aws.amazon.com/pricing/?nc2=h_ql_reinvent, 2015.
- [8] Amazon, "CloudWatch," <http://aws.amazon.com/whitepapers/>, 2015.
- [9] Amazon, "Crash of Amazon EC2 Cloud Services,," <http://www.businessinsider.com/amazon-lost-data-2011-4>, 2011.
- [10] M. Anala and G. Shobha, "Comparative study of application performance on virtual machine and physical machine," in *Computational Intelligence & Computing Research (ICCIC), 2012 IEEE International Conference on*, 2012, pp. 1-6.
- [11] M. Anand, "Cloud Monitor: Monitoring Applications in Cloud," *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on Communication, Networking & Broadcasting*, pp. 1-4, 2012.
- [12] S. Angeles, "Virtualization vs. Cloud Computing: What's the Difference?," <http://www.businessnewsdaily.com/5791-virtualization-vs-cloud-computing.html>, 2015.
- [13] Apache.org, "Apache Project," http://httpd.apache.org/ABOUT_APACHE.html, 2015.
- [14] D. Armstrong and K. Djemame, "Towards quality of service in the cloud," in *Proc. of the 25th UK Performance Engineering Workshop*, 2009.
- [15] J. Ashwini, C. Divya, and H. Sanjay, "Efficient resource selection framework to enable cloud for HPC applications," in *Computer and Communication Technology (ICCTT), 2013 4th International Conference on*, 2013, pp. 34-38.
- [16] L. Atzori, F. Granelli, and A. Pescapè, "A network-oriented survey and open issues in cloud computing," *Cloud computing: methodology, system, and applications, CRC, Taylor & Francis group*, 2011.

- [17] G. Aversano, M. Rak, and U. Villano, "The mOSAIC benchmarking framework: Development and execution of custom cloud benchmarks," *Scalable Computing: Practice and Experience*, vol. 14, 2013.
- [18] Azure.Microsoft, "Azure FC," <http://azure.microsoft.com/en-us/documentation/videos/fabric-controller-internals-building-and-updating-high-availability-apps/>, 2015.
- [19] G. Back, "Isolation, resource management and sharing in the KaffeOS Java runtime system," The University of Utah, 2002.
- [20] S. A. Baset, "Cloud SLAs: present and future," *ACM SIGOPS Operating Systems Review*, vol. 46, pp. 57-66, 2012.
- [21] I. Baumgart and B. Heep, "Fast but economical: A simulative comparison of structured peer-to-peer systems," in *Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on*, 2012, pp. 87-94.
- [22] H. A. Bheda and J. Lakhani, "QoS and performance optimization with VM provisioning approach in Cloud computing environment," in *Engineering (NUiCONE), 2012 Nirma University International Conference on*, 2012, pp. 1-5.
- [23] M. A. A. bin Mohd Shuhaimi, "The new services in Nagios: Network bandwidth utility, email notification and sms alert in improving the network performance," in *Information Assurance and Security (IAS), 2011 7th International Conference on*, 2011, pp. 86-91.
- [24] Bitnami.org, "Bitnami Cloud Images," http://bitnami.org/faq/cloud_amazon_ec2, 2012.
- [25] 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.
- [26] F. Brasileiro, F. Greve, M. Hurfin, J.-P. Le Narzul, and F. Tronel, "Eva: an event-based framework for developing specialised communication protocols," in *Network Computing and Applications, 2001. NCA 2001. IEEE International Symposium on*, 2001, pp. 108-119.
- [27] J. Brindza, J. Szweda, Q. Liao, Y. Jiang, and A. Striegel, "WiiLab: bringing together the Nintendo Wiimote and MATLAB," in *Frontiers in Education Conference, 2009. FIE'09. 39th IEEE*, 2009, pp. 1-6.
- [28] B. Cai, F. Xu, F. Ye, and W. Zhou, "Research and application of migrating legacy systems to the private cloud platform with cloudstack," in *Automation and Logistics (ICAL), 2012 IEEE International Conference on*, 2012, pp. 400-404.
- [29] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in *Parallel Processing (ICPP), 2011 International Conference on*, 2011, pp. 295-304.
- [30] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini, "Evaluating scheduling and replica optimisation strategies in OptorSim," in *Proceedings of the 4th International Workshop on Grid Computing*, 2003, p. 52.
- [31] Q. Cao, Z.-B. Wei, and W.-M. Gong, "An optimized algorithm for task scheduling based on activity based costing in cloud computing," in *Bioinformatics and Biomedical Engineering, 2009. ICBBE 2009. 3rd International Conference on*, 2009, pp. 1-3.

- [32] W. Cappelli and J. Kowall, "Magic Quadrant for Application Performance Monitoring," Technical report, UC Berkeley 2011.
- [33] E. Caron, L. Roderio-Merino, F. d. r. Desprez, and A. Muresan, "Auto-scaling, load balancing and monitoring in commercial and open-source clouds," 2012.
- [34] M. Castro, M. Costa, and A. Rowstron, "Should we build Gnutella on a structured overlay?," *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 131-136, 2004.
- [35] A. Chervenak and S. Bharathi, "Peer-to-peer approaches to grid resource discovery," in *Making Grids Work*, ed: Springer, 2008, pp. 59-76.
- [36] S. N. T.-c. Chiueh and S. Brook, "A survey on virtualization technologies," *RPE Report*, pp. 1-42, 2005.
- [37] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Roderio-Merino, L. M. Vaquero, *et al.*, "Monitoring service clouds in the future internet," *Towards the Future Internet-Emerging Trends from European Research*, pp. 1-12, 2010.
- [38] CLIQR, "CLIQR MAKES CLOUD BENCHMARKING AVAILABLE TO ALL USERS, PROVIDING CRITICAL INFORMATION NEEDED TO OPTIMIZE APPLICATION DEPLOYMENTS ON THE CLOUD,"
<http://www.cliqr.com/company/news-events/cliqr-makes-cloud-benchmarking-available-to-all-users-providing-critical-information-needed-to-optimize-application-deployments-on-the-cloud/>, 2015.
- [39] cloudharmony, "Azure Downtime," <https://cloudharmony.com/status-1year-of-storage-and-compute-group-by-regions-and-provider>, 2013.
- [40] CloudWatch, "CloudWatch,"
<http://awsdocs.s3.amazonaws.com/AmazonCloudWatch/latest/acw-dg.pdf>, 2014.
- [41] T. Crawford and R. Pettus, "A Unix command line argument processor," in *Southeastcon'88., IEEE Conference Proceedings*, 1988, pp. 484-487.
- [42] CSIRO, "CSIRO's ASKAP Radio Telescope,"
<http://www.atnf.csiro.au/projects/askap/index.html>, 2015.
- [43] K. Czajkowski, I. Foster, and C. Kesselman, "Agreement-based resource management," *Proceedings of the IEEE*, vol. 93, pp. 631-643, 2005.
- [44] K. Czajkowski, I. Foster, and C. Kesselman, "Co-allocation services for computational grids," in *Proc. 8th IEEE Symposium on High Performance Distributed Computing*, 1999.
- [45] F. Dabek, B. Zhao, P. Druschel, J. Kubiawicz, and I. Stoica, "Towards a common API for structured peer-to-peer overlays," in *Peer-to-Peer Systems II*, ed: Springer, 2003, pp. 33-44.
- [46] C. R. Davis, S. Neville, J. M. Fernandez, J.-M. Robert, and J. Mchugh, "Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures?," in *Computer Security-ESORICS 2008*, ed: Springer, 2008, pp. 461-480.
- [47] 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.
- [48] O. C. Derby, "FlexGP: a scalable system for factored learning in the cloud," Massachusetts Institute of Technology, 2013.

- [49] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010, pp. 27-33.
- [50] S. Distefano, A. Puliafito, M. Rak, S. Venticinque, U. Villano, A. Cuomo, *et al.*, "Qos management in cloud@ home infrastructures," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, 2011, pp. 190-197.
- [51] J. Dollner and K. Hinrichs, "Interactive, animated 3D widgets," in *Computer Graphics International, 1998. Proceedings*, 1998, pp. 278-286.
- [52] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for BPEL workflows using Amazon's elastic compute cloud," in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, 2009, pp. 140-147.
- [53] Eclipse, "What is Eclipse?," <http://help.eclipse.org/>, 2015.
- [54] A. C. Edwin and A. N. Madheswari, "Job Scheduling and VM Provisioning in Clouds," in *Advances in Computing and Communications (ICACC), 2013 Third International Conference on*, 2013, pp. 261-264.
- [55] M. A. El-Refaey and M. A. Rizkaa, "CloudGauge: a dynamic cloud and virtualization benchmarking suite," in *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, 2010, pp. 66-75.
- [56] V. C. Emeakaro, I. Brandic, M. Maurer, and I. Breskovic, "SLA-Aware application deployment and resource allocation in clouds," in *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, 2011, pp. 298-303.
- [57] A. FC, "Azure Configuration and APIs," http://snarfed.org/windows_azure_details#Configuration_and_APIs, 2015.
- [58] A. FC, "Azure Fabric Controller," <http://www.techopedia.com/definition/26433/azure-fabric-controller>, 2014.
- [59] M. Firdhous, S. Hassan, and O. Ghazali, "A Comprehensive Survey on Quality of Service Implementations in Cloud Computing," *International Journal of Scientific & Engineering Research*, vol. 4, pp. 118-123, 2013.
- [60] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun, "Benchmarking in the cloud: What it should, can, and cannot be," in *Selected Topics in Performance Evaluation and Benchmarking*, ed: Springer, 2013, pp. 173-188.
- [61] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler, "End-to-end quality of service for high-end applications," *Computer Communications*, vol. 27, pp. 1375-1388, 2004.
- [62] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1-10.
- [63] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid search schemes for unstructured peer-to-peer networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005, pp. 1526-1537.

- [64] S. V. Gogouvitis, V. Alexandrou, N. Mavrogeorgi, S. Koutsoutos, D. Kyriazis, and T. Varvarigou, "A monitoring mechanism for storage clouds," in *Cloud and Green Computing (CGC), 2012 Second International Conference on*, 2012, pp. 153-159.
- [65] 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.
- [66] Google, "Google App Engine," <https://cloud.google.com/appengine/>, 2014.
- [67] A. B. Grant and O. T. Eluwole, "Cloud resource management—Virtual machines competing for limited resources," in *AFRICON, 2013*, 2013, pp. 1-7.
- [68] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase, "Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration," in *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, 2006, p. 7.
- [69] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," *Security & privacy, IEEE*, vol. 9, pp. 50-57, 2011.
- [70] 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.
- [71] A. M. Hammadi and O. Hussain, "A framework for SLA assurance in cloud computing," in *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, 2012, pp. 393-398.
- [72] 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.
- [73] C. N. Hoefer and G. Karagiannis, "Taxonomy of cloud computing services," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, 2010, pp. 1345-1350.
- [74] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, 2009, pp. 101-111.
- [75] hyperic, "SIGAR," <https://support.hyperic.com>, 2015.
- [76] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 104-113.
- [77] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, *et al.*, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 159-168.
- [78] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *Communications Magazine, IEEE*, vol. 51, pp. 24-31, 2013.
- [79] C. Janssen, "PAYG," <http://www.techopedia.com/definition/26951/pay-as-you-go-payg>, 2015.
- [80] K. Junhom, S. Semkham, P. Lumlert, P. Niampoonthong, and V. Visoottiviseth, "Cloudbroid: An Android Mobile Application for CloudStack Management System,"

- in *Student Project Conference (ICT-ISPC), 2014 Third ICT International*, 2014, pp. 121-124.
- [81] S. Kailasam, N. Gnanasambandam, D. Janakiram, and N. Sharma, "Optimizing Service Level Agreements for Autonomic Cloud Bursting Schedulers," in *ICPP Workshops*, 2010, pp. 285-294.
 - [82] A. Kaur, L. Singh, and H. Singh, "STUDY OF PARAMETERS FOR EVALUATION OF SOFTWARE AS A SERVICE."
 - [83] J. Kirschnick, J. M. Alcaraz Calero, L. Wilcock, and N. Edwards, "Toward an architecture for the automated provisioning of cloud services," *Communications Magazine, IEEE*, vol. 48, pp. 124-131, 2010.
 - [84] A. Klein, C. Mannweiler, J. Schneider, and H. D. Schotten, "Access schemes for mobile cloud computing," in *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, 2010, pp. 387-392.
 - [85] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A Distributed Machine-learning System," in *CIDR*, 2013.
 - [86] D. Kreutz, A. Casimiro, and M. Pasin, "A trustworthy and resilient event broker for monitoring cloud infrastructures," in *Distributed applications and interoperable systems*, 2012, pp. 87-95.
 - [87] 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.
 - [88] K. Lai, M. Feldman, I. Stoica, and J. Chuang, "Incentives for cooperation in peer-to-peer networks," in *Workshop on economics of peer-to-peer systems*, 2003, pp. 1243-1248.
 - [89] J. Lakhani and P. Kumar, "Resource selection strategy based on propagation delay in Cloud," in *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, 2012, pp. 710-713.
 - [90] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven service request scheduling in clouds," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 15-24.
 - [91] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing the kazaa network," in *Internet Applications. WIAPP 2003. Proceedings. The Third IEEE Workshop on*, 2003, pp. 112-120.
 - [92] P. Leitner and J. Cito, "Patterns in the Chaos-a Study of Performance Variation and Predictability in Public IaaS Clouds," *arXiv preprint arXiv:1411.2429*, 2014.
 - [93] P. Leitner, Z. Rostyslav, A. Gambi, and S. Dustdar, "A framework and middleware for application-level cloud bursting on top of infrastructure-as-a-service clouds," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, 2013, pp. 163-170.
 - [94] 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.

- [95] P. Leung and S.-C. Cheung, "A CSCW framework for the flexible coupling of groupware widgets," in *Engineering of Complex Computer Systems, 1999. ICECCS'99. Fifth IEEE International Conference on*, 1999, pp. 9-20.
- [96] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 1-14.
- [97] D. Li, H. Liu, and A. Vasilakos, *An efficient, scalable and robust p2p overlay for autonomic communication*: Springer, 2009.
- [98] J. Li, Y. Xiong, X. Liu, and L. Zhang, "How Does Web Service API Evolution Affect Clients?," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*, 2013, pp. 300-307.
- [99] X.-Y. Li, L.-T. Zhou, Y. Shi, and Y. Guo, "A trusted computing environment model in cloud architecture," in *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, 2010, pp. 2843-2848.
- [100] J. Liang, R. Kumar, and K. Ross, "The kazaa overlay: A measurement study," in *Proceedings of the 19th ieee annual computer communications workshop*, 2004, pp. 17-20.
- [101] X. Liu, Y. Yang, D. Yuan, G. Zhang, W. Li, and D. Cao, "A generic QoS framework for cloud workflow systems," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, 2011, pp. 713-720.
- [102] LogicMonitor, "LogicMonitor " <http://www.logicmonitor.com/why-logicmonitor>., 2014.
- [103] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, pp. 716-727, 2012.
- [104] C. Luo, J. Zhan, Z. Jia, L. Wang, G. Lu, L. Zhang, *et al.*, "CloudRank-D: benchmarking and ranking cloud computing systems for data processing applications," *Frontiers of Computer Science*, vol. 6, pp. 347-362, 2012.
- [105] 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.
- [106] S. M. Magda, A. B. Rus, and V. Dobrota, "Nagios-based network management for Android, Windows and Fedora Core terminals using Net-SNMP agents," in *Roedunet International Conference (RoEduNet), 2013 11th*, 2013, pp. 1-6.
- [107] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—The business perspective," *Decision Support Systems*, vol. 51, pp. 176-189, 2011.
- [108] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, and M. Villari, "A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, 2011, pp. 1510-1517.
- [109] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST special publication*, vol. 800, p. 145, 2011.
- [110] Microsoft, "Azure " <http://azure.microsoft.com/en-in/overview/what-is-azure/>, 2015.
- [111] Microsoft, "Multi-Tenant Data Architecture," <https://msdn.microsoft.com>, 2015.

- [112] D. Milojević, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, pp. 0011-14, 2011.
- [113] S. Mohagheghi, J. Stoupis, and Z. Wang, "Communication protocols and networks for power systems-current status and future trends," in *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*, 2009, pp. 1-9.
- [114] A. Moniruzzaman, K. W. Nafi, and S. A. Hossain, "An experimental study of load balancing of OpenNebula open-source cloud computing platform," in *Informatics, Electronics & Vision (ICIEV), 2014 International Conference on*, 2014, pp. 1-6.
- [115] Monitis, "Monitis Monitoring Portal," <http://portal.monitis.com/>. 2014.
- [116] J. Moses, R. Iyer, R. Illikkal, S. Srinivasan, and K. Aisopos, "Shared resource monitoring and throughput optimization in cloud-computing datacenters," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 2011, pp. 1024-1033.
- [117] V. Muraleedharan, "Hawk-i HPC Cloud Benchmark Tool," *Msc in high performance computing, University of Edinburgh, Edinburgh*, 2012.
- [118] Nagios, "Nagios " <http://www.nagios.com.>, 2014.
- [119] nagios.org, "Nagios User Profiles," <http://users.nagios.org/>. 2014.
- [120] 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.
- [121] L. N. Nassif, J. M. Nogueira, and F. V. de Andrade, "Distributed resource selection in grid using decision theory," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, 2007, pp. 327-334.
- [122] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 237-250.
- [123] Nimsoft, "Nimsoft," <http://www.nimsoft.com/solutions/nimsoft-monitor/cloud.>, 2014.
- [124] NIST, "Security and Privacy Controls for Cloud-based Federal Information Systems," *Nat'l Inst. of Standards and Technology*, vol. 800-174, 2014.
- [125] D. Ocean, "SNMP," <https://www.digitalocean.com/community/tutorials/an-introduction-to-snmp-simple-network-management-protocol>, 2015.
- [126] OpenNebula.org, "OpenNebula Overview," <http://opennebula.org/documentation:rel4.0.>, 2014.
- [127] Oracle, "Cloud Infrastructure APIs and CLI," <http://docs.oracle.com/>, 2015.
- [128] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010, pp. 400-407.
- [129] H. A. Patel and A. D. Meniya, "A Survey on Commercial and Open Source Cloud Monitoring," *International Journal of Science and Modern Engineering (IJISME)*, ISSN, pp. 2319-6386, 2013.
- [130] J. F. Patterson, R. D. Hill, S. L. Rohall, and S. W. Meeks, "Rendezvous: An architecture for synchronous multi-user applications," in *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, 1990, pp. 317-328.

- [131] L. Pearlman, C. Kesselman, S. Gullapalli, B. Spencer Jr, J. Futrelle, K. Ricker, *et al.*, "Distributed hybrid earthquake engineering experiments: Experiences with a ground-shaking grid application," in *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, 2004, pp. 14-23.
- [132] 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.
- [133] D. Petcu, C. Craciun, M. Neagul, I. Lazcanotegui, and M. Rak, "Building an interoperability API for sky computing," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, 2011, pp. 405-411.
- [134] D. Petcu, C. Craciun, and M. Rak, "Towards a cross platform cloud API," in *1st International Conference on Cloud Computing and Services Science*, 2011, pp. 166-169.
- [135] M. Rak and G. Aversano, "Benchmarks in the cloud: The mosaic benchmarking framework," in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, 2012, pp. 415-422.
- [136] R. Ranjan and B. Benatallah, "Programming cloud resource orchestration framework: operations and research challenges," *arXiv preprint arXiv:1204.2204*, 2012.
- [137] R. Ranjan, L. Chan, A. Harwood, S. Karunasekera, and R. Buyya, "Decentralised resource discovery service for large scale federated grids," in *e-Science and Grid Computing, IEEE International Conference on*, 2007, pp. 379-387.
- [138] R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-peer-based resource discovery in global grids: a tutorial," *Communications Surveys & Tutorials, IEEE*, vol. 10, pp. 6-33, 2008.
- [139] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," *Cloud Computing*, pp. 195-217, 2010.
- [140] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network* vol. 31: ACM, 2001.
- [141] P. V. V. Reddy and L. Rajamani, "Evaluation of different hypervisors performance in the private cloud with SIGAR framework," *International Journal of Advanced Computer Science and Applications*, vol. 5, 2014.
- [142] RevealCloud, "The Cloud Advantage for Smaller Enterprises," <http://sandhill.com/article/>, 2014.
- [143] RevealCloud, "RevealCloud " <http://copperegg.com/>. 2014.
- [144] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, 2001, pp. 99-100.
- [145] L. Romano, D. De Mari, Z. Jerzak, and C. Fetzer, "A novel approach to QoS monitoring in the cloud," in *Data Compression, Communications and Processing (CCP), 2011 First International Conference on*, 2011, pp. 45-51.
- [146] M. Roseman and S. Greenberg, "GroupKit: A groupware toolkit for building real-time conferencing applications," in *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, 1992, pp. 43-50.
- [147] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*, 2001, pp. 329-350.

- [148] C. R. Rupakheti and D. Hou, "Evaluating forum discussions to inform the design of an API critic," in *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, 2012, pp. 53-62.
- [149] C. L. Sabharwal, "Java, java, java," *Potentials, IEEE*, vol. 17, pp. 33-37, 1998.
- [150] Salesforce, "CRM," <http://www.salesforce.com/>, 2013.
- [151] Salesforce.com., "Force.com Apex Code Developer's Guide.," "http://www.salesforce.com/us/developer/docs/apexcode/salesforce_apex_language_reference.pdf," 2013.
- [152] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, pp. 460-471, 2010.
- [153] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent, "SNMP traffic analysis: Approaches, tools, and first results," in *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, 2007, pp. 323-332.
- [154] M. Sellami, S. Yangui, M. Mohamed, and S. Tata, "PaaS-independent Provisioning and Management of Applications in the Cloud," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, 2013, pp. 693-700.
- [155] C. R. Senna, L. F. Bittencourt, and E. R. Madeira, "Service workflow monitoring in private clouds: The user point of view," in *Cloud Computing and Communications (LATIN CLOUD), 2012 IEEE Latin America Conference on*, 2012, pp. 25-30.
- [156] C. Services, "Cloudharmony " <https://cloudharmony.com/services>, 2014.
- [157] J. Shao, H. Wei, Q. Wang, and H. Mei, "A runtime model based monitoring approach for cloud," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 313-320.
- [158] P. Shivam, A. Demberel, P. Gunda, D. Irwin, L. Grit, A. Yumerefendi, *et al.*, "Automated and on-demand provisioning of virtual machines for database applications," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 1079-1081.
- [159] Y. Shu, L. Zhang, W. Zhao, H. Chen, and J. Luo, "P2P-based data system for the EAST experiment," *Nuclear Science, IEEE Transactions on*, vol. 53, pp. 694-699, 2006.
- [160] A. Solomon, D. Santamaria, and R. Lister, "Automated testing of unix command-line and scripting skills," in *Information Technology Based Higher Education and Training, 2006. ITHET'06. 7th International Conference on*, 2006, pp. 120-125.
- [161] SPAE, "Server Monitoring cloud," <http://www.rackaid.com/resources/server-monitoring-cloud>., 2014.
- [162] SPAE, "SPAE Features," http://shalb.com/en/spae/spae_features/, 2014.
- [163] J. Spring, "Monitoring Cloud Computing by Layer, Part 1," *Security & Privacy, IEEE*, vol. 9, pp. 66-68, 2011.
- [164] StackDriver, "Getting the Most out of CloudWatch," "https://storage.googleapis.com/stackdriverdotcom.appspot.com/Best_Practices_for_AWS_CloudWatch_Stackdriver_Ebook.pdf," 2014.

- [165] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, pp. 149-160, 2001.
- [166] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband internet performance: a view from the gateway," in *ACM SIGCOMM computer communication review*, 2011, pp. 134-145.
- [167] A. Takefusa, H. Nakada, T. Kudoh, and Y. Tanaka, "An advance reservation-based co-allocation algorithm for distributed computers and network bandwidth on qos-guaranteed grids," in *Job Scheduling Strategies for Parallel Processing*, 2010, pp. 16-34.
- [168] A. Tchana, B. Dillenseger, N. De Palma, X. Etchevers, J.-M. Vincent, N. Salmi, *et al.*, "Self-scalable benchmarking as a service with automatic saturation detection," in *Middleware 2013*, ed: Springer, 2013, pp. 389-404.
- [169] R. Tolosana-Calasanz, J. Á. Bañares, C. Pham, and O. F. Rana, "Enforcing qos in scientific workflow systems enacted over cloud infrastructures," *Journal of Computer and System Sciences*, vol. 78, pp. 1300-1315, 2012.
- [170] A. Turner, A. Fox, J. Payne, and H. S. Kim, "C-mart: Benchmarking the cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, pp. 1256-1266, 2013.
- [171] S. Vijayakumar, Q. Zhu, and G. Agrawal, "Automated and dynamic application accuracy management and resource provisioning in a cloud environment," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010, pp. 33-40.
- [172] F. Wang and W. Du, "A test automation framework based on web," in *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, 2012, pp. 683-687.
- [173] W. Wang and M. W. Godfrey, "Detecting API usage obstacles: A study of iOS and Android developer questions," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, 2013, pp. 61-64.
- [174] E. Wibowo, "Cloud management and automation," in *Rural Information & Communication Technology and Electric-Vehicle Technology (rICT & ICeV-T), 2013 Joint International Conference on*, 2013, pp. 1-4.
- [175] wikipedia.org, "Resource management (computing)," [https://en.wikipedia.org/wiki/Resource_management_\(computing\)](https://en.wikipedia.org/wiki/Resource_management_(computing)), 2015.
- [176] L. Wonham, "Compuware APM," <http://www.websitemagazine.com/content/blogs/posts/archive/2011/05/30/compuware-gomez-introduce-new-apm-solution.aspx>, 2011.
- [177] C.-H. Wu, "Differentiated admission for peer-to-peer systems: incentivizing peers to contribute their resources," 2003.
- [178] B. Yang and H. Garcia-Molina, "Comparing hybrid peer-to-peer systems," in *Proceedings of the 27th Intl. Conf. on Very Large Data Bases*, 2001.
- [179] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "C-meter: A framework for performance analysis of computing clouds," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 472-477.

- [180] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1-10.
- [181] S. Zaman and D. Grosu, "Combinatorial auction-based dynamic vm provisioning and allocation in clouds," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 107-114.
- [182] L.-J. Zhang and Q. Zhou, "CCOA: Cloud computing open architecture," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, 2009, pp. 607-616.
- [183] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, and P. Strazdins, "Investigating decision support techniques for automating cloud service selection," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, 2012, pp. 759-764.
- [184] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7-18, 2010.
- [185] 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.
- [186] B. Y. Zhao, J. Kubiawicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," 2001.
- [187] Q. Zheng, H. Chen, Y. Wang, J. Duan, and Z. Huang, "Cosbench: A benchmark tool for cloud object storage services," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 998-999.
- [188] S. Zhou, G. Rogers, M. Hogan, S. Ardon, T. Hu, and A. Seneviratne, "An Incentive based routing Algorithm for improving message forwarding in structured Peer-to-Peer Networks," in *Wireless Broadband and Ultra Wideband Communications, 2007. AusWireless 2007. The 2nd International Conference on*, 2007, pp. 58-58.
- [189] D. Zisis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, pp. 583-592, 2012.

Appendix A: CLAMBS Prototype Implementation Files

A.1: CLAMBS Monitoring

A.1.1: clambs monitoring agent sigar functions

```

public class SigarHelper {

    private static Sigar sigar = new Sigar();

    public static void getInformationsAboutMemory() {

        Mem mem = null;
        try {
            mem = sigar.getMem();
        } catch (SigarException se) {
            se.printStackTrace();
        }

        System.out.println("Actual total free system memory: "
            + mem.getActualFree() / 1024 / 1024+ " MB");
        System.out.println("Actual total used system memory: "
            + mem.getActualUsed() / 1024 / 1024 + " MB");
        System.out.println("Total free system memory .....: " + mem.getFree()
            / 1024 / 1024+ " MB");
        long trymem = mem.getFree() / 1024 / 1024;
        System.out.println("Total free system memory .....:"+trymem);
        System.out.println("System Random Access Memory....: " + mem.getRam()
            + " MB");
        System.out.println("Total system memory.....: " + mem.getTotal()
            / 1024 / 1024+ " MB");
        System.out.println("Total used system memory.....: " + mem.getUsed()
            / 1024 / 1024+ " MB");

    }

    public static long getProcessId(String procName) throws SigarException
    {
        ProcessFinder find=new ProcessFinder(sigar);
        long pid=find.findSingleProcess("State.Name.eq="+procName);
        ProcMem memory=new ProcMem();
        memory.gather(sigar, pid);
        return pid;
    }

    public static void main(String[] args) throws Exception{

        getInformationsAboutMemory();
        long pid = getProcessId("eclipse");
        System.out.println("Pid: "+pid);

    }
}

```

```
}
```

A.1.2 CLAMBS MONITORING AGENT SNMP FUNCTIONS

```
public SNMPAgent(Map args)
{
    //initialize the agent from AgentConfig.properties
    this.configFile = (String)((List)args.get("c")).get(0);
    this.bootCounterFile = new File((String)((List)args.get("bc")).get(0));
    this.server = new DefaultMOServer();
    MOServer[] moServers = new MOServer[] { server }; //MOServer for managed objects
    //read AgentConfig.properties
    InputStream configInputStream =

        SNMPAgent.class.getResourceAsStream("AgentConfig.properties");
        if (args.containsKey("cfg")) {
            try {
configInputStream = new
FileInputStream((String) ArgumentParser.getValue(args, "cfg", 0));
            }
            catch (FileNotFoundException ex1) {
                ex1.printStackTrace();
            }
        }
        final Properties props = new Properties();
        try {
            props.load(configInputStream);
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
        MOInputFactory configurationFactory = new MOInputFactory() {
            public MOInput createMOInput() {
                return new PropertyMOInput(props, SNMPAgent.this);
            }
        };
        //read AgentTableSizeLimits.properties
        InputStream tableSizeLimitsInputStream =

        SNMPAgent.class.getResourceAsStream("AgentTableSizeLimits.properties");
        if (args.containsKey("ts")) {
            try {
                tableSizeLimitsInputStream =
                    new FileInputStream((String) Argument-
Parser.getValue(args, "ts", 0));
            }
            catch (FileNotFoundException ex1) {
                ex1.printStackTrace();
            }
        }
        tableSizeLimits = new Properties();
        try {
```

```

        tableSizeLimits.load(tableSizeLimitsInputStream);
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
    //MessageDispatcher for Message between agent and manager
    MessageDispatcher messageDispatcher = new MessageDispatcherImpl();
    addListenAddresses(messageDispatcher, (List)args.get("address"));
    agent = new AgentConfigManager(new OctetString(MPv3.createLocalEngineID()),
        messageDispatcher,
        null,
        moServers,
        ThreadPool.create("SNMPAgent", 500),
        configurationFactory,
        new DefaultMOPersistenceProvider(moServers,
            configFile),
        new EngineBootsCounterFile(bootCounterFile));
    }
    //add a ListenAddresses with given MessageDispatcher and a list of addresses
    protected void addListenAddresses(MessageDispatcher md, List addresses) {
        for (Iterator it = addresses.iterator(); it.hasNext();) {
            Address address = GenericAddress.parse((String)it.next());
            TransportMapping tm =
                TransportMap-
pings.getInstance().createTransportMapping(address);
            if (tm != null) {
                md.addTransportMapping(tm);
            }
        }
    }

    public void run() {

        agent.initialize();

        registerMIBs();
        agent.setupProxyForwarder();
        agent.setTableSizeLimits(tableSizeLimits);
        agent.run();
    }

```

A.2: CLAMBS WORK LOAD GENERATOR

A.2.1: SQL WORK LOAD GENERATOR

```

public class SQL extends Thread{

    BOOLEAN SETUPFORMFILE = TRUE;

    public void run(){
        if(setupformfile){

```

```

        StandardJMeterEngine jmeter = new StandardJMeterEngine();

        // Initialize Properties, logging, locale, etc.
JMeterUtils.loadJMeterProperties("C:\\Users\\Khalid\\Downloads\\ZIPs\\jakarta-
jmeter-2.5.1\\bin\\jmeter.properties");
JMeterUtils.setJMeterHome("C:/Users/Khalid/Downloads/ZIPs/jakarta-jmeter-2.5.1");

JMeterUtils.initLogging();
JMeterUtils.initLocale();

// Initialize JMeter SaveService
try {
    SaveService.loadProperties();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

// Load existing .jmx Test Plan
FileInputStream in;
try {
    in = new FileInputStream("SQL_Request.jmx");
    HashTree testPlanTree = SaveService.loadTree(in);
    in.close();

    SampleResult sampleResult = new SampleResult();
    testPlanTree.add("sampleResult", sampleResult);
    // Run JMeter Test
    jmeter.configure(testPlanTree);
    jmeter.run();
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

System.out.println("Type of result will be : Thread ID,Sample Time(ms),JDBC Re-
quest,code,Status,Thread Group ,type return,Bytes,Latency");
BufferedReader br;
try {
    br = new BufferedReader(new FileReader("SQL_result.jtl"));
    String line;
    int result = 1;
    while ((line = br.readLine()) != null) {
        // process the line.
        System.out.println("Result of you QUERY "+result+": "+line);
        result++;
    }
    br.close();
} catch (FileNotFoundException e) {

```

```

        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    Path path = Paths.get("SQL_result.jtl");
    try {
        Files.delete(path);
    } catch (NoSuchFileException x) {
        System.err.format("%s: no such" + " file or directory%n", path);
    } catch (DirectoryNotEmptyException x) {
        System.err.format("%s not empty%n", path);
    } catch (IOException x) {
        // File permission problems are caught here.
        System.err.println(x);
    }

    }else {
        // Engine
        StandardJMeterEngine jm = new StandardJMeterEngine();
        // jmeter.properties
        JMeterU-
        tils.LoadJMeterProperties("C:\\Users\\Khalid\\Downloads\\ZIPs\\jakarta-jmeter-
        2.5.1\\bin\\jmeter.properties");

        HashTree hashTree = new HashTree();

        // HTTP Sampler
        JDBCSampler sqlSampler = new JDBCSampler();
        sqlSampler.setName("VN running");
        sqlSampler.setVariableNames("MYSQL");
        sqlSampler.setQuery("select * from net_qos.net_qos");
        sqlSampler.setQueryType("Select Statement");
        ConstantTimer timer = new ConstantTimer();
        timer.setDelay("300");
        sqlSampler.addTestElement(timer);

        DataSourceElement confi = new DataSourceElement();
        confi.setName("MYSQL");
        confi.setUsername("root");
        confi.setPassword("1234");
        confi.setDbUrl("jdbc:mysql://localhost:3306/net_qos");
        confi.setDriver("com.mysql.jdbc.Driver");

        // Loop Controller
        TestElement loopCtrl = new LoopController();
        ((LoopController)loopCtrl).setLoops(100);
        ((LoopController)loopCtrl).addTestElement(sqlSampler);
        ((LoopController)loopCtrl).setFirst(true);

        // Thread Group

```

```

        SetupThreadGroup threadGroup = new SetupThreadGroup();
        threadGroup.setNumThreads(1);
        threadGroup.setRampUp(1);
        threadGroup.setSamplerController((LoopController)loopCtrl);

        // Test plan
        TestPlan testPlan = new TestPlan("MY TEST PLAN");
        //testPlan.setTestPlanClasspath("E:/workspace/jmeter/test.jmx");

        SampleResult sampleResult = new SampleResult();
        //sampleResult.getLatency();

        hashTree.add("testPlan", testPlan);
        hashTree.add("loopCtrl", loopCtrl);
        hashTree.add("threadGroup", threadGroup);
        hashTree.add("jdbcSampler", sqlSampler);
        hashTree.add("sampleResult", sampleResult);

        jm.configure(hashTree);

        jm.run();
    }
}
}

```

A.2.2: HTTP WORK LOAD GENERATOR

```

public class HTTP extends Thread{

    @Override
    public void run(){

        // Engine
        StandardJMeterEngine jm = new StandardJMeterEngine();
        JMeterUtils.loadJMeterProperties("C:\\Users\\Khalid\\Downloads\\ZIPs\\jakarta-
jmeter-2.5.1\\bin\\jmeter.properties");
        JMeterUtils.setJMeterHome("../jakarta-jmeter-2.5.1");
        JMeterUtils.initLocale();

        HashTree hashTree = new HashTree();

        // HTTP Sampler
        HTTPSampler httpSampler = new HTTPSampler();
        httpSampler.setDomain("www.unsw.edu.au");
        httpSampler.setPort(80);
        httpSampler.setPath("/");
        httpSampler.setMethod("GET");
    }
}

```

```

// Loop Controller
TestElement loopCtrl = new LoopController();
((LoopController)loopCtrl).setLoops(3);
((LoopController)loopCtrl).addTestElement(httpSampler);
((LoopController)loopCtrl).setFirst(true);

// Thread Group
SetupThreadGroup threadGroup = new SetupThreadGroup();
threadGroup.setNumThreads(1);
threadGroup.setRampUp(1);
threadGroup.setSamplerController((LoopController)loopCtrl);

// Test plan
TestPlan testPlan = new TestPlan("MY TEST PLAN");

SampleResult sampleResult = new SampleResult();
sampleResult.getLatency();

hashTree.add("testPlan", testPlan);
hashTree.add("loopCtrl", loopCtrl);
hashTree.add("threadGroup", threadGroup);
hashTree.add("httpSampler", httpSampler);
hashTree.add("sampleResult", sampleResult);

jm.configure(hashTree);

jm.run();

}

}

```

A.3: CLAMBS DATABASE CONNECTION

```

public class MySQLConn {
    public Connection conn = null;
    public Statement stmt = null;
    public ResultSet rs = null;
    public int ChangedRow = 0;
    public MySQLConn()
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance();

            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/net_qos", "root", "1234");
        }
        catch(Exception e)
        {

```

```

        System.out.println("DataBase Connect Failed! "+e);
    }

}

public static void main(String args[])
{
    MySQLConn dbconn=new MySQLConn();
}

public Statement getConn()
{
    return stmt;
}

public ResultSet getDBrs(String sql)//select
{
    try
    {
        rs=stmt.executeQuery(sql);
    }
    catch (Exception e) {
        System.out.println("(select)Exception Catcher:<br/>" +e);
        e.printStackTrace();
    }
    return rs;
}

public boolean exec(String sql)//exce a sql statement
{
    boolean flag=false;
    try
    {
        stmt = conn.prepareStatement(sql);
        flag=stmt.execute(sql);
    }
    catch (Exception e) {
        System.out.println("(execc)Exception Catcher:<br/>" +e);
    }
    return flag;
}

public int setDBupdate(String sql)
{
    try
    {
        ChangedRow=stmt.executeUpdate(sql);
    }
    catch (Exception e) {
        System.out.println("(update)Exception Catcher:<br/>" +e);
    }
    return ChangedRow;
}

public void setDBclose() //close

```

```

{
    try
    {
        if(rs!=null)
        {
            rs.close();
            rs=null;
        }
        if(stmt!=null)
        {
            stmt.close();
            stmt=null;
        }
        if(conn!=null)
        {
            conn.close();
            conn=null;
        }
    }
    catch(Exception e)
    {
        System.out.println("(Close DataBase)Exception Catch-
er:<br/>" + e);
    }
}

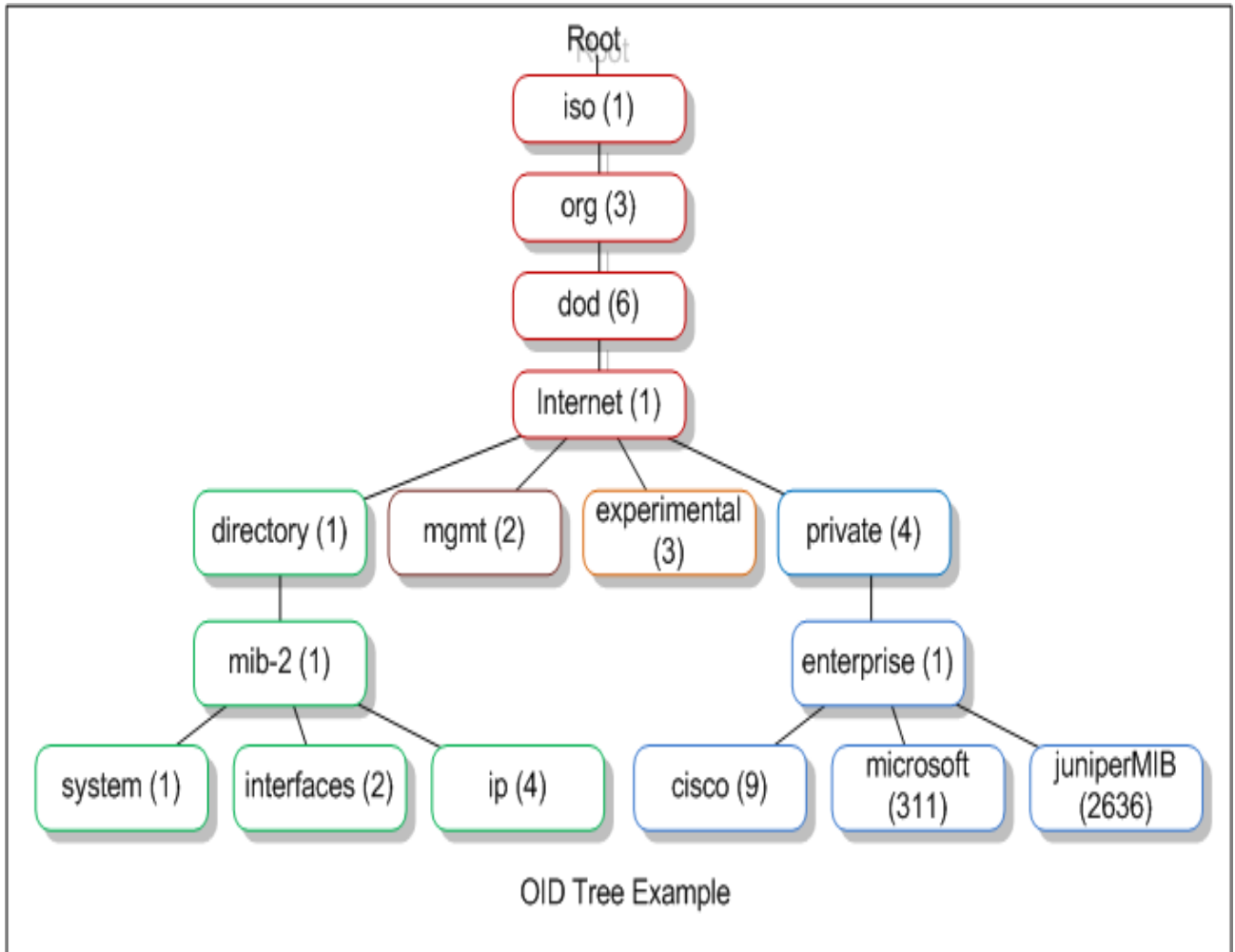
public void setDBStatementClose()//close the statement after up-
date,delete,or insert
{
    try
    {
        stmt.close();
    }
    catch (Exception e) {
        System.out.println("Exception Catcher:<br/>" + e);
    }
}

public int getRowCount(ResultSet rs)//count of result
{
    int RowCount=0;
    try {
        rs.last();
        RowCount = rs.getRow();
        rs.beforeFirst();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return RowCount;
}

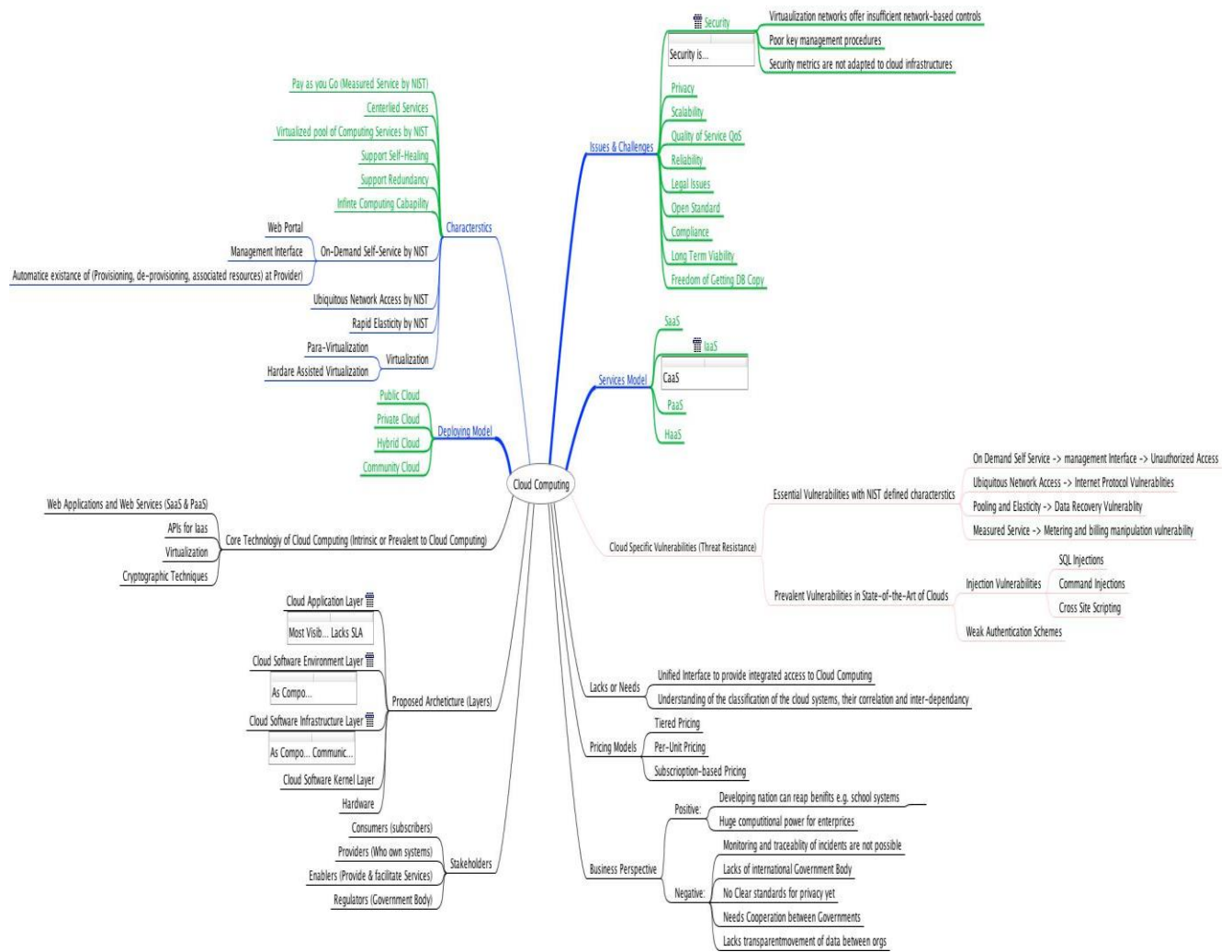
}

```

Appendix B: SNMP MIBs Tree



Appendix C: Brainstorming Mind Map



Glossary

QoS: Quality of Service

SLA: Service Level Agreement

Multi-clouds: multiple cloud platforms

Cross-layers: across multiple cloud platform layers

SaaS: refers to the model in which applications are provided as a hosted service to cloud customers who access these services via the Internet.

PaaS: is a cloud computing model to provide applications' components over the Internet. PaaS delivers hardware and software tools mostly these tools are required for applications development.

IaaS: provides access to fundamental compute, storage, and network resources in a virtualized environment.

VM: Virtual Machine

VMM: Virtual Machine Monitor

PAYG: Pay As You Go model is a utility computing billing method which is applied in cloud computing

EC2: Elastic computing platform provided by Amazon.com

SIGAR: System Information Gatherer and Reporter

SNMP: Simple Network Management Protocol

MIBs: Management Information Base

RESTful: REST stands for Representational State Transfer

FC: Fabric Controller of Azure platform

JVM: Java Virtual Machine