A Vendor Neutral QoS Monitoring Model for SaaS Cloud Computing Solutions

Frankline Makokha
School of Computing and Informatics
University of Nairobi
Nairobi, Kenya
goldmedalist321 [at] gmail.com

Elisha, E. Opiyo
School of Computing and Informatics
University of Nairobi
Nairobi, Kenya

Abstract— **The increased uptake in use of cloud computing solutions and an upsurge in the number of cloud service providers has raised the need for cloud consumers to validate the QoS derived from the various cloud solution providers. This validation requires use of cloud QoS monitoring tools that are developed by entities other than the cloud providers themselves. Further, the tool should not be tied to the underlying architecture of any cloud platform to make it usable across various cloud vendors, thus making it vendor neutral. Due to unavailability of vendor neural QoS monitoring tools, cloud users have had to rely on the tools developed by the same providers from whom they get the cloud services. This is due to the fact that the tools are dependent on the underlying cloud architecture of the specific cloud vendor. In cases where the client has more that one cloud provider for the same services, it is not possible to compare the level of QoS derived from the different providers since the tools are not portable. This paper presents a model for developing a cloud QoS monitoring tool for SaaS cloud computing solutions that is not tied to an underlying architecture of any particular cloud provider.**

Keywords- Quality of Service, Vendor Neutral, Cloud, Cloud Computing, Model, Browser Extension, Web

I. INTRODUCTION

With the advancement in cloud computing technology, more and more companies are opting to adopt this technology due to lower cost of investment compared to actual purchase of hardware and software systems. With this trend by most companies, more and more cloud service providers are coming up, leading to competition for available clients.

To facilitate the client in choosing which cloud service provider offers better services, there has to be a mechanism through which clients can gauge the Quality of Service (QoS) offered by the various cloud providers.

The various cloud QoS monitoring models in existence, namely the Agent Based Model, Adaptive QoS-driven Monitoring Model, CloudQual, The Quality of Service MONitoring as a Service Model (QoSMONaaS) are all tied to the physical infrastructure of the service provider and hence a monitoring tool developed using any of the above model can not be used across multiple cloud vendors [1]. This means the current cloud QoS monitoring tools are not portable across various cloud vendors.

Cloud service models can be divided into three main categories: SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service), [2].

The above listed models can be depicted architecturally as in the figure 1.

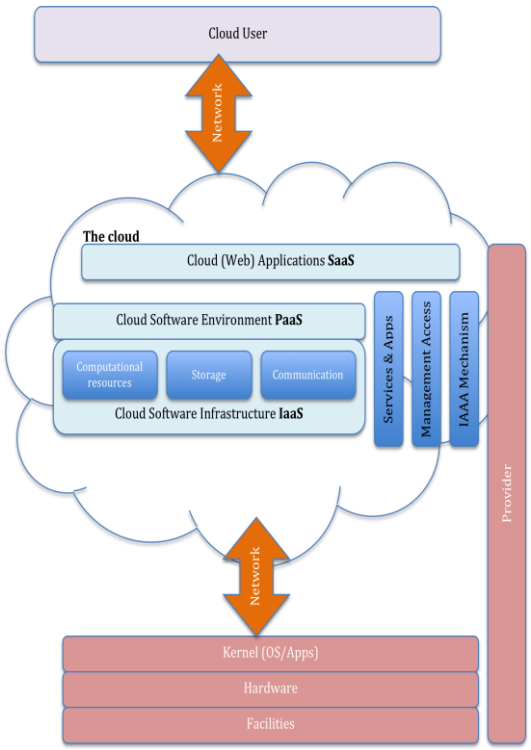

Figure 1: The Cloud Reference Architecture
Adapted from [3]

II. HIGH LEVEL ARCHITECTURE FOR CLOUD QOS MONITORING MODEL

From the analysis done by [1], the current high-level architecture of the existing cloud QoS monitoring models is as shown in figure 2.

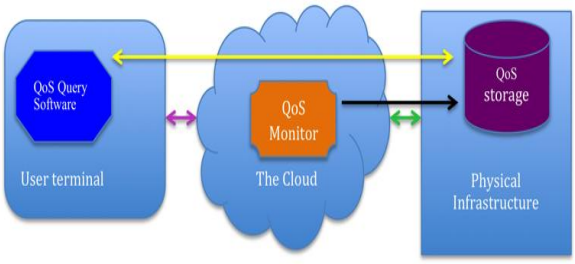

Figure 2: High Level Architecture of the Current QoS Monitoring Model

From the above model, the QoS monitoring tools developed reside in the cloud, they measure the QoS experienced by the user and store the results in the providers systems for querying by the user. This in deed poses the possibility of vendor bias since the service provider and the QoS tool developer are the same entity, the provider further stores the results before the user can query them. In cases where the Service Level Agreements (SLA) is strict, trust issues on the measured QoS may arise.

Further, from the above diagram, it is clear that the tool is tied to the architecture of the cloud on which it runs. This implies that the tool cannot be used on any other cloud platform and thus in case the service user wishes to compare the QoS levels of different providers of the same service, it would not be possible using the same tool.

To eliminate possible cases of vendor bias, it would be wise to develop a model that is not tied to the system architecture any specific cloud provider. Further, the QoS results as measured should be directly relayed to the service user without first being stored on the cloud provider's infrastructure.

The possible high-level architecture for the solution to the above problem is depicted in figure 3.

Figure 3: A Conceptual Architecture of a Possible Vendor Neutral QoS Monitoring Model.

From figure 3, the tool resides on the user's terminal, monitors the QoS of the cloud service, and stores the results in the user's terminal.

The solution to this puzzle lies in the access method to the service. The three cloud service models, namely SaaS, PaaS and IaaS, can be accessed by the user via two methods, namely, a cloud provider specialized application installed on the user terminal or via the web browser. The various access methods to the three cloud service models are as shown in figure 4.

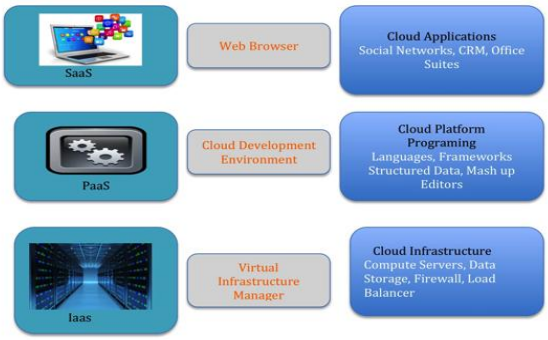

Figure 4 : Coud Services Access Methods
Adapted from [4].

A close analysis of this access mode shows that the cloud provider specialized application access method is still dependant on the cloud provider and is thus vendor specific. The only access method that is vendor neutral is the browser based method. As [5] puts it, Services provided by SaaS are accessed by end users through Web portals.

This method grants an opportunity for developing a vendor neutral model for monitoring the QoS of cloud solutions. This will involve an in-depth study of the architecture of web browsers for proper understanding of the various components that comprise the browser to inform on how a third party tool can be incorporated in the browser to extend its functionality to include cloud QoS monitoring.

III. The Web Browsers Architecture

A web browser is an essential application program for accessing the Internet [6]. It is defined by [7] as a program that can read and fetch documents locally as well as from sites around the world via the Internet. An alternative definition by [8] is that it is a program that retrieves documents on the World Wide Web from remote servers and displays them on screen, either within the browser window itself or by passing the document to an external helper application. Based on these definitions a web browser can be defined as an end user application with an interface (graphical/text based) through which the user can interact with content on the Internet and the World Wide Web by specifying the Uniform Resource Identifiers of the content.

A reference architecture for a web browser is as depicted in figure 5.

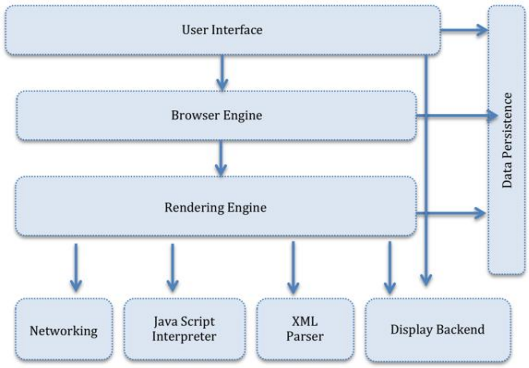

Figure 5: A Reference Architecture for a Web Browser. Adapted from [8]

A reference architecture for a domain captures the fundamental subsystems common to systems of that domain, as well as the relationships between these subsystems [8].

A. Web Browser Sub Components

The functions of the above listed sub components are as highlighted below [8]:

The User Interface subsystem: is the layer between the user and the Browser Engine. It provides features such as toolbars, visual page-load progress, smart download handling, preferences, and printing. It may be integrated with the desktop environment to provide browser session management or communication with other desktop applications.

The Browser Engine subsystem is an embeddable component that provides a high level interface to the Rendering Engine. It loads a given URI and supports primitive browsing actions such as forward, back, and reload. It provides hooks for viewing various aspects of the browsing session such as current page load progress and JavaScript alerts. It also allows the querying and manipulation of Rendering Engine settings.

The Rendering Engine subsystem translates a URI into a visual representation. It is capable of displaying HTML and XML documents, optionally styled with Cascading Style Sheets (CSS), as well as embedded content such as images. It is responsible for page layout and may contain reflow algorithms, which incrementally adjust the position of elements on the page. This subsystem also includes the HTML parser.

The Networking subsystem implements file transfer protocols such as HTTP and FTP. It translates between different character sets, and resolves MIME media types for files. It may include a cache of recently retrieved resources.

The JavaScript Interpreter evaluates JavaScript (also known as ECMAScript) code, which may be embedded in web pages. JavaScript is an object-oriented scripting language developed by Netscape. Certain JavaScript functionality, such as the opening of popup windows, may be disabled by the Browser Engine or Rendering Engine for security purposes.

The XML Parser subsystem parses XML documents into a Document Object Model (DOM) tree. This is one of the most reusable subsystems in the architecture. Most browser implementations leverage an existing XML Parser, rather than rewriting their own from scratch.

The Display Backend subsystem provides drawing and windowing primitives, a set of user interface widgets, and a set of fonts. It may be tied closely with the Operating System.

The Data Persistence subsystem stores various data associated with the browsing session on disk namely high level data such as bookmarks or toolbar locations and lower level data such as cookies, cache and security certificates

B. Browser Extensibility

An extensible system is one that permits later revision of the previously designed base system: additions to, improvements upon, or replacements of existing functionality [9]. Modern day browsers have three ways of improving their functionality, namely vide extensions, plug-ins or widgets.

In computing, a plug-in (or add-in / addin, plugin, extension or add-on / addon) is a software component that adds a specific feature to an existing software application [10].

Plugins allow browsers to parse and display content that is not traditional HTML [11]. Webpages that depend on plugins, directly invoke them through the use of appropriately set <object> and <embed> tags.

Browser extensions are meant to extend or modify the default behavior of a browser and make use of well-defined APIs provided to them by browsers [11].

A clear difference between the terms add-on, plug-in, widget and extension is highlighted in table1.

Table 1: Key Differences between Add-on, Plug-in, Widget and Extension and other related terms.

No	Term	Description	Examples
1.	Browser Plug in	Third party downloadable piece of code that enables browsers to render content that is not HTML	a) Adobe flash player b) Java plugin c) QuickTime player d) Google talk e) Macromedia flash
2.	Browser Widget	Plug in with a visible interface that can be dragged and dropped	a) Calendar b) Weather
3.	Extension	Third party downloadable piece of code meant to increase or improve the functionality of a web browser	a) Google docs offline b) Gmail c) Chrome we store payments d) Translate
4.	Add-on	A generic term for a plug-in, extension or widget	
5.	Patch	A piece of code designed to update a computer program or its supporting data and operating system, to fix it or improve it	

A web browser architectural diagram, with an add-on is as shown in figure 6.

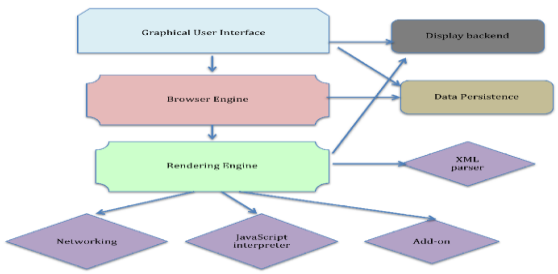

Figure 6: Browser architecture with an add-on sub component.
Adapted from [12]

The Generic structure on how an add-on interfaces with an existing program is a shown in figure 7.

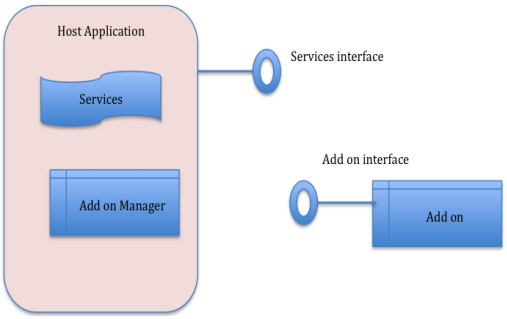

Figure 7: A generic interface between an add-on and a host application.
Adapted from [10]

From the descriptions provided in table 1, a vendor neutral model for cloud QoS monitoring is best designed and developed as an Extension.

IV. The Architecture of a Browser Extension

The generic structure of a web browser extension is as depicted in figure 8.

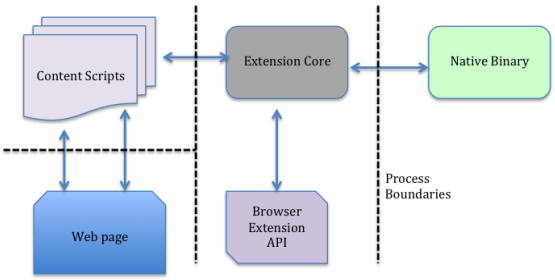

Figure 8 : The Architecture of a Browser Extension
Adapted from [13].

A more zoomed-in view of the architecture of a browser extension is as shown in figure 9.

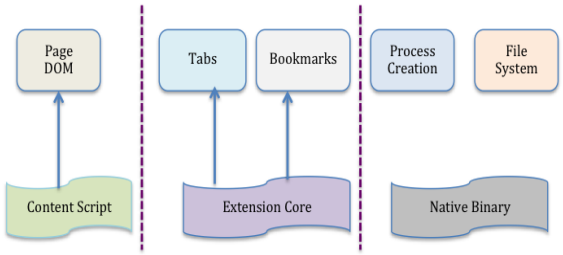

Figure 9: A zoomed in view of a browser extension. Adapted from [14].

From figure 9, the Content scripts are limited to only interacting with untrusted Web content and therefore execute with no privileges; the Extension core implements extension specific features including browser User Interface (UI) modification, interacting with system level resources via browser's extension (Application Programming Interface) API and therefore executes with the extension's full privileges; while the native binary code interacts with the host machine.

One example of a browser, Chrome, separates privileges between different components of an extension [15]. In particular, the content script of an extension can directly interact with web contents. However, by default it does not have the permissions to access browser modules, except that it can communicate to the extension core via postMessage.

The extension core has most assigned privileges, but it is insulated from web pages. It has to use content scripts or invoke XMLHttpRequest to communicate with the web content. The native binary of an extension, running as an NPAPI plugin, has the most privileges as it can run arbitrary code or access any files.
This privilege separation with a modular architecture was introduced in modern browsers to address the security challenges of legacy monolithic browsers where extension code and code interacting with Web page content execute in a unified JavaScript heap.

V. Proposed Vendor Neutral Cloud QoS Monitoring Model

To develop a vendor neutral model for cloud QoS monitoring, the model will have to be developed as an extension, which would be pegged to a particular browser. A high level architecture, of this proposed model would be as shown in figure 10.

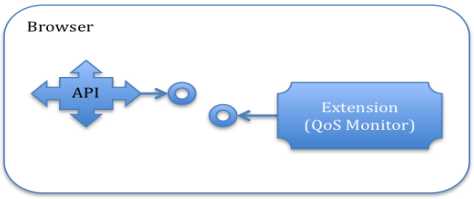

Figure 10 : High Level architecture of the Proposed Vendor Neutral QoS Monitoring Model.

A zoomed in view of the Extension sub component would be as depicted in figure 11.

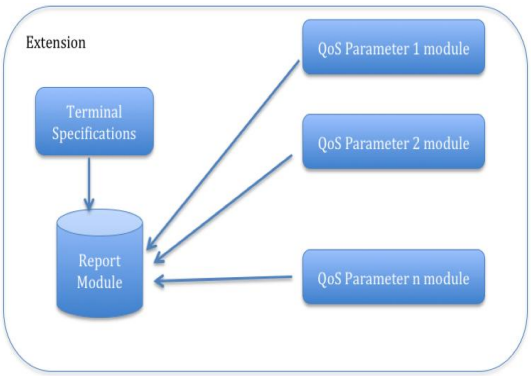

Figure 11 : A Zoomed in View of the Vendor Neutral Model.

The terminal specifications module is to retrieve the specifications of the system (user equipment) on which the extension is running, e.g. RAM capacity, CPU speed and the Internet speeds at the time of monitoring. This is important in cases where the QoS from the cloud is also affected by the terminal that measured those QoS. The QoS parameter module monitors the specific parameter it is programed to monitor and stores the results in the report module.

A. Implementation of the Proposed Vendor Neutral Cloud QoS Model

The proposed vendor neutral model can be implemented as an extension for any of the web browsers. The development tools in this model can be a combination of any of the standard web development technologies, namely HTML, CSS, JavaScript and Canvas. For data storage and rendering, any of the database technologies namely SQL and MySQL could be used subject to what the web browser APIs support.

CONCLUSION

With the growth of public cloud offerings, for cloud customers it has become increasingly difficult to decide which provider can fulfill their QoS requirements, since each cloud provider offers similar services at different prices and performance levels with different set of features [16]. Due to the difficulty in portability of the existing cloud QoS monitoring models, the proposed Vendor Neutral model will be handy to cloud users to validate the QoS data as retrieved from the cloud providers system and also compare the performance of two or more cloud providers offering the same service.

ACKNOWLEDGMENTS

We acknowledge the assistance and guidance provided by the late Prof.Okello-Odongo, of the University of Nairobi, during the nascent stages of this research before his untimely demise.

REFERENCES

[1] Makokha, F., Opiyo, E. and Okello-odongo (2017). Challenges of Quality of Service Monitoring in Cloud Computing Solutions. *International Journal of Computer and Information Technology* Vol. 06 Issue 06.

[2] Gorelek, E. (2013). *Cloud Computing Models*. MEng, Massachusetts Institute of Technology (MIT).

[3] Kumar, S. and Goudar, R. H. (2012) Cloud Computing – Research Issues, Challenges, Architecture, Platforms and Applications: A Survey. *International Journal of Future Computer and Communication*.Vol. 1, No. 4.

[4] Ashraf, I. (2014). An Overview of Service Models of Cloud Computing. *International Journal of Multidisciplinary and Current Research*. Vol.2 (July/Aug 2014 issue.

[5] Buyya, R., Broberg, J. and Goscinski, A. (Eds., 2011). *Cloud Computing: Principles and Paradigms*. Hoboken, New Jersey: John Wiley & Sons, Inc.

[6] Junghoon, O., Seungbong, L., and Sangjin L. (2011). Advanced evidence collection and analysis of web browser activity: *Proceedings of The Digital Forensic Research Conference*, DFRWS 2011 USA, New Orleans, LA (Aug 1st - 3rd).

[7] Vetter, R. J., Spell, C., and Ward, C. (1994). Mosaic and the World Wide Web. *IEEE Computer*, Volume: 27, Issue: 10.

[8] Grosskurth, A. and Godfrey, M. (2005). A Reference Architecture for Web Browsers. *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference Software Maintenance*, Budapest, Hungary, IEEE Computer Society, Los Vaqueros Circle, CA.

[9] Lerner, B. S. (2011). *Designing for Extensibility and Planning for Conflict: Experiments in Web-Browser Design*. PhD., University of Washington.

[10] Jain, J. (2015) *Security Plug-ins Handbook: A Student's Guide*, Chicago: InfoSec Institute.

[11] Starov, O. and Nikiforakis, N.(2017) "XHOUND: Quantifying the Fingerprintability of Browser Extensions," *2017 IEEE Symposium on Security and Privacy* (SP), San Jose, CA.

[12] Vrbanec, T., Kirić, N. and Varga, V. (2013) The evolution of web browser architecture, in M Mokrys, S Badura, A Lieskovsky (Eds.), *SCIENCOF 2013 :Proceedings of The 1st International Virtual Scientific Conference*. Publishing Society: Slovakia:

[13] Karim, R. (2015) *Techniques And Tools For Secure Web Browser Extension Development*. PhD, The State University of New Jersey.

[14] Barth, A., Porter Felt, A., Saxena, P. and Boodman, A. (2010), Protecting Browsers from Extension Vulnerabilities: *Proceedings of the 17th Network and Distributed Systems Security Symposium*, San Diego, CA (Feb 28th – March 3rd, 2010).

[15] Liu, L., Zhang, X., Yan, G and Chen, S. (2012) Chrome Extensions: Threat Analysis and Countermeasures, NDSS '12: *Proceedings of the 19th Network and Distributed System Security Symposium*. San Diego, California February 5-8, 2012.

[16] Mamoun, H. M and Ibrahim, E. M. (2014). A Proposed Framework for Ranking and Reservation of Cloud Services. *International Journal of Engineering and Technology*, Volume 4, No. 9.