Towards Dynamic QoS Monitoring in Service Oriented Architectures

Norman Ahmed[1,2] and Bharat Bhargava[1]

[1]*Department of Computer Science, Purdue University, 305 N. University St., W. Lafayette, IN 47906, U.S.A.*
[2]*Air Force Research Laboratory/RIS, 525 Brooks Rd, Rome, NY 13441, U.S.A.*

Keywords: Service Oriented Architecture, Web Services, Aspect Oriented Programming, Web Services Security, Cloud Computing, Quality of Service.

Abstract: Service Oriented Architecture (SOA) is an architectural style that provides agility to align technical solutions to modular business Web Services (WS) that are well decoupled from their consumers. This agility is established by interconnecting WS family of standards specification protocols (commonly referred to as WS-* (WS-star)) to enable security, ease of service interoperability and orchestration complexities when extending services across organizational boundaries. While orchestrating services or chaining services in varying ways to satisfy different business needs, on highly scalable cloud platforms is undeniably useful, it is increasingly challenging to effectively monitor Quality of Service (QoS), especially, service response time. This is due to a) lack of proper formulation of the WS-star interconnections mechanisms, and b) the transient performance behaviour intrinsic to the heterogeneity of the hardware and shared virtualized network and IO resources built on the cloud platforms. We present an analysis of WS-star standards, classifying and discussing their inter-dependencies to provide a basis for QoS monitoring context on protocol formulation. We then illustrate a practical implementation of a dynamic QoS monitoring mechanism using runtime service instrumentation with Aspect Oriented Programming (AOP). Preliminary evaluations show the efficiency of computing QoS on a transient performance cloud platform.

1 INTRODUCTION

Service Oriented Architecture (SOA) is the architectural style that provides agility to align technical solutions to modular business Web Services (WS) that are well decoupled from their consumers in the cloud environment. Built on a Virtualization of heterogeneous hardware and software stack on a SOA-based architecture as its technical foundation, cloud computing is a computing model that enables socio-economic benefits due to its on demand computing resource availability.

In this computing model, service providers and consumers are typically decoupled by means of common universal registries known as Universal Description Discovery and Integration (UDDI) and mediation mechanisms. Service capabilities, interface options, Quality of Service (QoS), and security constraints are described in the Service Level Agreement (SLA) (Overton, 2002) that is typically published in the UDDI.

The SLA document represents a contractual agreement for obligating the service provider to comply both functional and non-functional parameters of the registered service. The non-functional parameters are QoS attributes, such as service response time and service up time (i.e. 95%-99.999%) that are not known by the consumers before runtime (Erl, 2005) nor by the provider when orchestrating variable services to satisfy different business needs.

To ease the interoperability complexity and security concerns, especially for web services, SOA encourages the use of WS-* standardized specification, referred to WS-star. The forefronts of these protocol specifications are the ones used for data transport (i.e. SOAP/HTTP(s), *WS_Security, and WS_SecureConversation*) message level security. Typically, services are developed and deployed by multiple software designers and system integrators without prior knowledge of their effective protocol interconnections when service are orchestrated, the process of chaining services in various ways to satisfy different business needs.

Due to the magnitude of the available standards, chained services have higher chance of overlapping

Ahmed N. and Bhargava B..
Towards Dynamic QoS Monitoring in Service Oriented Architectures.
DOI: 10.5220/0005460001630171
In *Proceedings of the 5th International Conference on Cloud Computing and Services Science* (CLOSER-2015), pages 163-171
ISBN: 978-989-758-104-5

some functionality, especially security functionalities, that hinder the overall QoS advertised in the published SLA. Coupled with the transient performance behaviour inherent in cloud platforms, further complicates this mixture of standard-based design and contractual compliance requirements to guarantee QoS.

Consider a realistic scenario where two or more orchestrated services deployed in the cloud that implement *WS_Security* to enforce encryption and digital signatures for both inbound and outbound traffic. The overall response time across the chain will be highly impacted due to the potential security functionalities overlap across the services. The main reason is that each service performs encryption and digital signature, which is typically a performance hog. One alternative solution in this case is the use of *WS_SecureConversation*. However, detecting such overlap is increasingly challenging due to the nature of these services' development and deployment by multiple teams in different times. Typically, a Business Process Execution Language (BPEL) is used during orchestration to either determine response time by waiting till response is received or configure it with a proper timeout. Note that these response time evaluations are statically performed in nature.

In addition, there is transient variable performance behaviour of the clouds' VM network and IO interfaces due to multi tenant resource sharing (Mei, et. Al. 2013). For example, over 300 million test cases conducted on nine cloud providers over seven days (Alistair, 2011) have shown *performance time-of-the-day variability* in virtualized environments. Later studies (Zhonghong, 2012) showed such transient performance behaviour is due to the hardware heterogeneity that the cloud is built of. Therefore, it is prudent to dynamically uncover QoS friendly alternatives at runtime to improve service response time, thus, the main objective of our work.

There is a large array of research that addresses WS performance issues; to name a few, some QoS monitoring research have been designed around service selection (Fung, 2005), (Tian, 2004) composition (Mietzner, 2010), (Fung 2005), and dynamic soft QoS guaranteeing (Abdelzaher, 1999). An area that has been substantially overlooked and poorly studied is the understanding of the underlying WS-* standard specification behaviour under the cloud, especially, regarding service response time for web services.

In this work, we propose a dynamic QoS monitoring scheme on SOA-based services on virtualized shared cloud platforms. The goal is to capture the improper protocol formulation and the underlying platform performance variations to effectively compute service response time without any modification to the service code to improve hard QoS guaranteeing on virtualized environments.

In this paper, we present analysis of WS-* (WS-star) by classifying and discussing their interdependencies to show QoS impacts on improper protocol formulation. We then illustrate dynamic QoS monitoring mechanisms in a widely adopted service container (JBoss). Thus, our contribution is two fold:

- We developed an effective scheme for dynamically monitoring orchestrated services and computing service response time in cloud environments without service code modification or recompilation.
- While the proposed instrumentation scheme is designed for QoS monitoring, it can also be used to detect malicious service in the chain, simply, by instrumenting the method calls that reach beyond its intended service end point.

The rest of the paper is organized as follows. Section 2 gives a brief overview of SOA ecosystem with especial emphasis on web services. We then discuss WS-security protocols and their interdependencies in section 3. We show our proposed approach in section 4 followed by the implementation and experimentation to illustrate the effectiveness of our approach in section 5 and the related work in section 6. Finally, section 7 provides the conclusion and future work.

2 SOA ECO SYSTEM

SOA is an architectural style that promotes a high degree of service decoupling and rapid system development and deployment that span across traditional organization boundaries. The traditional SOA triangle paradigm consists of a service registry (i.e. UDDI), a service provider and a consumer as depicted in Figure 1.

Figure 1: SOA Triangle System Model.

At a high level, web service (WS) is an approach of building web accessible services where the service

providers publish/register their service in the UDDI registry and the service consumers discover and invoke it. The two wide spread paradigms for building services compliant with WS protocols are a Representational State Transfer (REST) (Erl, 2012), referred to RESTful services, and the Simple Object Access Protocol (SOAP)-based services (discussed next).

WS are built on standard specifications to facilitate their integration and secure execution. The core of the WS architecture (WS Architecture, 2002) outlines a set of service characterization that enables these complex functionalities to co-exist. However, the actual specifications of the standards have been collaborated and authored by many organizations such as; W3C, OASIS, OMG, IBM, Microsoft, Oracle, and xmlsoap.org, which makes difficult to fully realize the goals of their interactions.

There has been a considerable research effort that addresses the magnitude of the available standards, their cross-referencing design and development difficulties. For example, in (Gamble, 2011), authors proposed a Security Meta-Language for guiding the formulation of secure messages in WS architecture that model the security relevant portions of the standard for their consistent, comprehensive, and correct applications.

Others have addressed this through the use of enterprise-level integration (i.e. Apache Camel), meditation (i.e. Enterprise Service Bus), and Orchestration (i.e. BPEL) tools. However, dynamically monitoring these critical protocol functionalities over transient performance platforms has not been sufficiently addressed in these tools and in a generic fashion.

2.1 RESTful Services

The RESTful Services paradigm is a lightweight service implementation scheme that avoids preserving service state and the use of the underlying message level security. In other words, the traditional encryption and digital signatures are not employed in this service model due to its computational and bandwidth requirements. RESTful services are stateless services where responding in a timely manner to every service request is critical, thereby widely used in non-critical applications such as; gmail access, facebook updates, amazon consumer interactions, etc.

A transport security layer (TSL) or SSL over HTTP (https) is typically used to secure RESTful services. Such security solutions are sufficient for point-to-point connection oriented where a service call is authenticated and securely responds to the request. However, this point-to-point security solutions are ill suited in orchestrated/chained service interactions where a service request from a consumer has to reach out to other services in which these services further reach other services in the chain that are possibly in different domains in order to respond to such request.

To remedy these limitations, the use of message-level security is introduced in the standard protocols such as: *WS_Security, WS_SecureConversation*, and *WS_Policy*. The key idea of message level security is to structure and wrap the message (both the request and response) by sealing it in an envelope (SOAP) and associating it with security attributes (saml token) to safeguard its access and on transit.

2.2 SOAP-based Services

SOAP-based services provide granular message level security using WS-* family of protocols in which WS-Security is at the forefront. Cryptographic and digital signature techniques are the core of protecting SOAP messages from attacks. As a consequent, this introduces a performance overhead to the services (Liu, 2005). As the services are orchestrated, these performance overheads increase in the order of magnitude due to the overlaps of the security functionalities. Detecting these overlaps of such critical security functionalities to improve QoS is the focus of our work.

In order to effectively illustrate the interconnection of the performance-degrading protocol formulation and avoid hiding the concept in a myriad of protocol standards, we limit our protocol interdependency analysis (discussed next) to only those protocols that impact QoS, specificaly policy enforcement and message level security protocols, confidentiality and integrity.

3 WEB SERVICE PROTOCOL INTERDEPENDENCIES

WS decoupling is typically achieved by means of common registries known as Universal Description and Discovery Integration (UDDI). Services deployed in UDDI are discoverable through either WS Application Language (WADL) or WS Description Language (WSDL) standard specifications as depicted in Figure 2 (top left box), and access control protocols (bottom left box).

WADL and WSDL are the two defacto standards for defining web service capabilities. These include: service URI, services, security capabilities, and QoS attributes using *WS_PolicyAttachment* for encryption, signatures, policies, and *WS-Addressing* for end point service response delivery. Discovery and access control protocols have no impact on QoS, therefore, in this work; we only give special emphasis on confidentiality and integrity protocols.

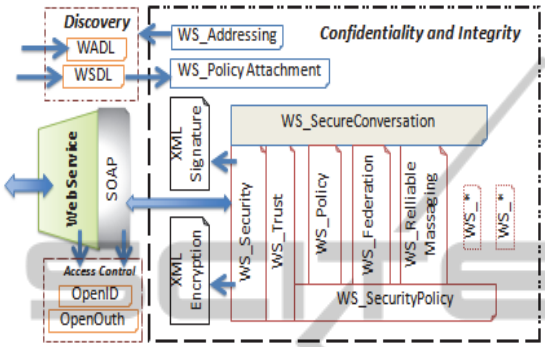

Figure 2: Anatomy of End-to-End Web Service Security Protocols - Service Discovery (top left box), Access Control (lower left box), and Confidentiality and Integrity for message level security (right box).

WS-Security is the core of WS-star protocol for confidentiality and integrity of the service. The WS-Security standard describes the security attributes of service and task delegation between services to facilitate secure authentication, authorization and invocations. Each new security concept or interface specification defined in WS-Security brings additional WS-* family of standards which play a significant role in expressing a web service's security posture.

For example, bridging communication between secure environments require protocols to specify cross-domain access controls. The Security Assertion Markup Language (SAML) provides the authentication and authorization among and across services, even in different security domains (Oasis-open, 2007), and eXtensible Access Control Language (XACML) provides the security policy enforcements for the authorizations that cross the organizational boundaries (Oasis-open, 2012).

Further, WS-Trust is required to broker authentication information, however, WS-Trust does not describe the security functionality of services and its capacity to fulfill the security needs. Instead, it delegates to *WS-SecurityPolicy* to describe the security policy which in turn uses WS-Policy. WS-Policy exchanges policy decisions and enforcement capabilities for every request, introducing more latency for QoS constraint services, especially if such capabilities deployed in a remote service domains.

In addition, WS-Security defines XML-Signature and XML-Encryption standards for digital signatures and encryption of XML documents to ensure the integrity of the exchanged SOAP messages/envelope. The more security capabilities added the more standard protocols needed. Thus, the SOAP message size increases, which consequently require more bandwidth and computationally intensive operations in encryption, signature, and verifications in which contribute to other QoS issues, especially when services are deployed across cloud domains or consumers with resource constrained devices (mobile).

QoS violations are imminent when improper protocol formulation is coupled with the transient performance behaviour of the underlying platform. A recent study (Zhonghong, 2012) shows that the virtualized heterogeneous hardware built on the cloud has performance variations that can reach up 60% between instances. Thus, dynamically intercepting and monitoring orchestrated services on such platforms are crucial in order to improve QoS guarantees and consequently prevent SLA violations.

4 SYSTEM MODEL

A motivating example of cross-domain service orchestration scenario is depicted in Figure 3 below.

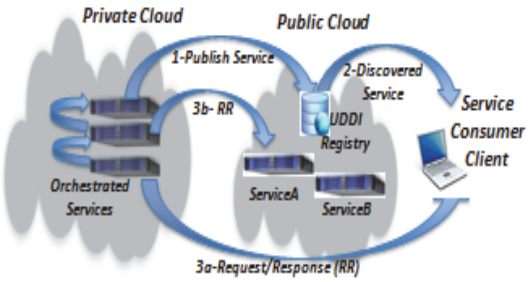

Figure 3: High-level architecture for orchestrated services across private and public clouds.

The high-level architecture above depicts a typical orchestrated service deployment across security domains, public and private. The top arrows marked (1-Publish Service) and (2-Discovered Service) show the service registration flow to the UDDI by the service provider where then the consumer client (depicted as the laptop) discovers that service. The client invokes that service as shown by the arrow-

marked (3a-Request/Response (RR)) and gets back a response. The arrow marked (3b-RR) between the clouds show the cross-domain service invocation that jointly satisfy the consumers' request. Note that all arrows represent a bio-directional data flow.

Chaining services in such environments is typically configured using BPEL. However, once services are deployed in the service container or application servers (eg. Jboss, Oracle, IBM Glassfish, etc.), these configurations are static, and thereby, fail to adapt to the changes of the underlying cloud platforms. Computing service response time in such setting typically requires reconfiguring or even re-designing the services.

We approached this problem by deploying interceptors in the service containers to seamlessly collect service information at runtime and compute response times while considering the transient behaviour of the deployed cloud platform. Service information can then be analyzed by machine learning to predict future QoS attributes, dynamically update SLA in the registry, or even migrate services instances to cloud platforms that are experiencing less performance issues in different regions. In this work, we focus on the detection scheme only.

4.1 QoS Criteria

There are several non-functional QoS metrics categories and service performance attributes in SOA-based WS. In this work, we only consider WS performance, specifically service response time for orchestrated services on cloud platforms.

4.2 Approach Overview

Most QoS attributes in SOA are not a one-size-fit-all for all consumer requests. A priori knowledge of any given QoS attribute for the prospective consumer is difficult to predict (Erl, 2005). Several QoS monitoring approaches offered solutions that improve QoS over the years. However, none have addressed the impact of the overlapping security protocols due to their criticality of the service protection coupled with the performance variability of the underlying cloud platforms.

The basic idea of our approach is to non-intrusively instrument services without introducing overhead. Our design is based on two steps, detection and aggregation. We use Aspect Oriented Programming (AOP), a dynamic application instrumentation framework first introduced in (Kiczales, 1997). AOP allows service code instrumentation without modifications or recompilation of the code. The instrumented data collected/detected at runtime from each service is then forwarded to the QoS auditor web service (referred as QAudit) to aggregate and then compute response time.

4.3 Service Instrumentation with AOP

Typically, collecting accurate QoS information at runtime is achieved by inserting general purpose logging statements in pre-compile time and during service composition. QoS metrics can then be based on the aggregate of these logs. Such techniques are inefficient and ill suited in cloud computing platforms due to the performance variability behaviour that are not under the control of the service provider. Since accurate QoS attributes cannot be predicted during service registration, dynamic service instrumentation is critical.

We achieve such dynamicity with AOP. A basic AOP model defines two instrumentation primitives known as *pointcut designators* (PCD) and *advice*. The *PCD's* are typically points in the program where inserting instrumentation is not too hard, for example, method calls are very often used as one of the fundamental *PCD*. These PCD's are simple instrumentation primitives that can gather critical information without any modifications to the code.

On the other hand, the *advice* is the point where an aspect to be instrumented can be weaved in. The result of PCD and advice generated will then forwarded to externally configured component, in our case, QAudit. QAudit web service evaluates the best QoS metrics under that given cloud platforms performance behaviour or overlapping security functionalities on the services, in which the service provider can take any action necessary such as; either update the SLA for the prospective users, reconfigure security protocols or project future QoS metrics of the given time of the day.

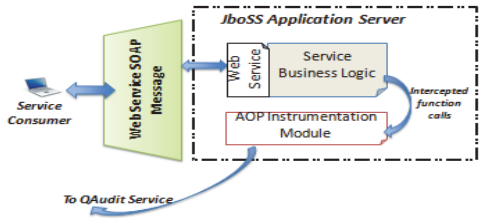

Figure 4: Service Anatomy and AOP Instrumentation Module inside the Jboss service container or app server.

AOP enables user level service interception capabilities within application servers. As depicted

in Figure 4, we used an AOP plugin as a module in the Jboss Application server where our services are deployed, known as JbossAOP (JbossAOP, 2003). JbossAOP instruments services deployed in the service container by intercepting the execution of all aspects of the program, such as specific object on the program, a function parameter values, or method calls within or across program calls.

The performance overhead of the AOP depends on the knowledge of the application (Alexanderson, 2010). For QoS monitoring, the overhead is proportion to the number of the interception points within the services. To limit such overhead, we only intercept WS-Security related function calls, specifically, prior and post encryption, and signature operations in which are negligible when tested in public cloud environment as shown in our previous work (Azarmi, et. al. 2012). Note that one can also instrument communication methods if needed to uncover rogue/compromised service reaching outside its intended endpoints.

4.4 QoS Auditor Web-service

As depicted in service anatomy diagram in Figure 4 above, the service container enables hooks to instrument the services' business logic where the instrumented data can then be sent to the listening service, QoS Auditor (*QAudit*) web service. The QAudit receives the pre and post *WS_Security* function call timing information collected from the diverse orchestrated services under the current performance of the services' environment (VM's). For example, some services are deployed in cloud platforms that are built on different hardware, hypervisor, and possibly running *VM* migration and load balancing algorithms by the cloud provider to accommodate between the tenants.

4.5 QoS Monitoring in Orchestrated Services

The WS Business Process Execution Language (WSBPEL) defines the orchestration of WS standard language for service chaining and execution. Identifying performance bottlenecks in orchestrated services from multiple providers within BPEL engines is a challenging task given the dynamicity of the cloud platforms that's not known a priori.

As described in the previous section, AOP instruments services deployed in the service container by intercepting the execution of all aspects of the program (i.e. method calls) across program calls. Since orchestrated services are also program calls across domains, AOP can effectively intercept orchestrated WS. We will describe our implementation approach in the next section.

5 IMPLEMENTATION AND EXPERIMENTATION

We are interested in computing service response time for secure web services orchestrated across cloud platforms (public/private) as illustrated in the high-level architecture in Figure 3. In this section, we discuss our prototype and show the preliminary evaluations on private cloud deployments, and the proposed QoS computations scheme.

5.1 Experimental Setup

Our experimental cloud platform uses a private cloud built on *OpenStack*, a cloud management software stack, on a cluster of 4 machines (Dell Z400) with Intel Xeon 3.2 GHz Quad-Core with 8GB of memory. At a high-level, *OpenStack* consists of a controller and computing management applications. We divided our four machines into one controller node and 3 compute nodes. As the name implies, the controller node is to simplify cloud platform management by enabling on demand elasticity, i.e., provision/de-provisioning *VM* instances, adding/removing hardware and instantly making it available in the computing resource pool.

The three compute nodes allow us provisioning 20 virtual CPU's (vCPU) in which we assigned 10 small *VM* instances, 2 vCPU per instance for service deployments. We used a total of 10 VMs with Ubuntu Linux for service consumer (clients) and secure services in all of our experiments.

5.2 Implementation

We developed a CXF-based secure web services *(WS_Security* and *WS_SecureConversation* enabled*)* and deployed in Jboss application server. The integration of AOP with Jboss container was done using JbossAOP (JbossAOP, 2003) library, a pluggable user specified instrumentation module for Jboss application servers. We leveraged AspectJ (AspectJ, 2001), a stand-alone Java implementation of AOP, as the service instrumentation algorithms for intercepting the WS-Security method calls.

5.3 Preliminary Results

It has been previously reported that performance difference between *WS-Security and WS_SecureConversation* in web services are in the order of magnitude higher in *WS-Security* (Liu, et.al. 2005). To illustrate in the context of service orchestration under the transient performance behaviour of the cloud, we configured three secure services that implement *WS_Security* with 3 others that implement *WS_SecureConversation* with the same logic, a secure weather report, deployed in a Linux virtual machines described above.

We orchestrated the 3 services with different configurations while assuring the security of the service. Service configuration is application and domain specific, thus, to illustrate the basic concept; we chained and invoked services in the following format:

$Req^1 \leftarrow\rightarrow S^{ws} \leftarrow\rightarrow S^{ws} \leftarrow\rightarrow S^{ws} \leftarrow\rightarrow S^{ws} \leftarrow\rightarrow ...$
$Req^2 \leftarrow\rightarrow S^{ws} \leftarrow\rightarrow S^{sc} \leftarrow\rightarrow S^{ws} \leftarrow\rightarrow S^{sc} \leftarrow\rightarrow ...$
$Req^n \leftarrow\rightarrow ...$

The requests Req^n interacts with service S^{ws} implemented with *WS_Security* and then S^{sc} implemented with *WS_SecureConversation* and so on.

To mimic the performance variability of the cloud platforms, services requests and responses were performed while the system is running *cpu* and memory intensive applications. We observed the system performance using the built in Ubuntu system monitor (krell) showing a load over 50%-70% usage of the memory and cpu. The service response times received by *QAudit* service, when aggregated, ranged between microseconds to seconds; thus, clearly show QoS impacts on security function overlaps.

These observations show the non-intrusive way of computing QoS in cloud platforms. However, the actual results may vary depending on the service logic and other factors when expended into the public cloud deployments, thereby, considering it in our future work.

6 RELATED WORK

To the best of our knowledge, there is no in-depth analysis of WS-star protocol formulation in SOA in the context of QoS monitoring that reflect the transient performance behaviour of the underlying cloud platforms. Thus, we divide our related work section into two parts; we first discuss works in WS-* performance improvements and next we provide QoS Management tools and techniques that are relevant to our work.

6.1 WS*- Performance Improvements

There are large volumes of research that employ different methods to address performance improvements on web-services. To name a few: SOAP header envelope reduction techniques, efficient XML parsing and compression methods, and binary and canonicalization techniques.

With the rise of business heterogeneity, orchestrated services pose further callings for selecting and complying with an accurate advertised QoS attributes, especially service response time. As these schemes have set the foundation of WS performance improvement, our approach was inspired by such mechanisms and further extended to dynamically monitor orchestrated services in a virtualized environment.

6.2 QoS Management

QoS management can be classified into three categories: resource allocation, service composition, monitoring and fine-tuning QoS parameters within the services. In this work, we focused on the latter two. It's intuitive to see that an effective resource sharing can aid QoS guarantees; moreover, service composition or selection also plays a critical role in such guarantees.

To highlight some studies in this category, early works, such as (Abdelzaher and Shin, 1999), proposed a virtual service that enables the selection of multiple deployed concrete services depending on the users' QoS interest. A set of cooperative autonomous agents that enable optimal web service composition is proposed in (Brahmi, 2013). Within the context of service selection, similar to QAudit approach, Q-Peer (Li, et.al. 2007), a distributed QoS registry is proposed to monitor and collect information on running services to assist consumers for the reliability of the service where as we focus on service response time improvements.

It has been noted that the inaccuracy and violations of QoS in various papers and spurred a wide range of research approaches, to name a few; QoS verifications during service registration (Abdelzaher and Shin, 1999), extending UDDI functionalities (ShaikhAli et. al, 2003), introducing new protocol languages to define SLA (Lamanna, 2003), SLA template adjustments (Spillner and Schill, 2009) and new frameworks for dynamic service monitoring and selections in a realistic

environment (Tian, et. al, 2004). Along the lines of the WS protocol research, a modification of WS-Agreement protocol to enable dynamic run-time renegotiation and SLA adjustments to guarantee QoS when SLA violation is expected to occur is proposed in (Modica, et. al, 2007).

All of the above approaches face adaptability challenges due to the proposed changes required in the protocol standards. Our work can accurately and non-intrusively detects the transient behaviour of the cloud platforms to prevent SLA violations without modifying the service code or the standard protocols. Furthermore, our work will complement the works of fine-tuning QoS parameters for efficient service composition, selection and monitoring schemes to maximize QoS and prevent SLA violations.

7 CONCLUSIONS

Guaranteeing hard QoS on orchestrated web-services in SOA and virtualized cloud platforms are increasingly challenging due to security critical functionality overlaps and the transient performance behaviour of such platforms. In this paper, we developed an effective mechanism to dynamically monitor orchestrated services and compute service response tme while considering the underlying performance behaviour of the cloud platforms.

We implemented our proposed approach with Aspect Oriented programming (AOP) and illustrated with a practical scenario to validate our design using three secure services deployed in a private cloud. In our future work, we consider experimental traces over periods of time in our private with public (i.e. Amazon) cloud instances deployed in different geographic locations.

ACKNOWLEDGEMENTS

Authors would like to thank Jim Hanna at AFRL for setting up the experimental platform, and special thanks to the reviewers for their valuable feedback that made this paper more readable.

REFERENCES

Abdelzaher, T. F., & Shin, K. G., 1999. QoS Provisioning with qContracts in Web and Multimedia Servers. *In the 20th IEEE Real-Time Systems Symposium*.

Alexanderson, R., Ohman, P., and Karlson, J., 2010. Aspect Oriented Implementation of Fault Tolerance: An assessment Overhead. *In Computer Safety, Reliability, and Security*. Lecture Notes in Computer Science, Volume 6351, pp 466-479.

Alistair C., 2011. Cloud Performance From the Users Prospective. http://www.bitcurrent.com/download/cloud-performance-from-the-end-user-perspective/.

AspectJ, 2001. http://eclipse.org/aspectj/

Azarmi, M., Angin, P., Bhargava, B., Ahmed, N., et al., 2012. End-to-End Security in Service Oriented Architecture, *In* SRDS12, *the 31st IEEE Int. Symposium on Reliable Distributed Systems*.

Brahmi, Z., 2013. QoS-aware Automatic Web Service Composition based on Cooperative Agents. *In WETICE, The 22nd IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.*

Erl, T., 2005. *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall.

Erl T., et al, 2012. *SOA with REST*, Prentice Hall. 1st ed.

Fung, C. et al., 2005. A Study of Service Composition with QoS Management. *In ICWS'05, IEEE International Conference on Web Services.*

Gamble, R. and Baird, R., 2011. Developing Security Meta-language Framework. *In Proceedings of the 44th Hawaii Int. Conference on System Sciences.*

JbossAOP, 2003. http://www.jboss.org/jbossaop.

Kiczales, et al., 1997. Aspect-Oriented Programming. *In ECOOP'97, Object-Oriented Programming, lecture Notes in CS. Vol. 1241, pp. 220–242.*

Lamanna, D., Skene, J., and Emmerich, W., 2003. SLAng: A Language for Service Level Agreements. In the 9th IEEE Workshop on Future Trends of Distributed Computing Systems. FTDCS.

Li, F., et al., 2007. Q-Peer: A Decentralized QoS Registry Architecture for Web Services. In ICSOC'07, *International Conference on Service Oriented Computing.* LNCS 4749,pp.145-156.

Liu, H., Pallikara, S., and Fox, G., 2005. Performance of Web Service Security. *In Proceedings of the 13th Annual Mardi Gras Conference.*

Mei, Y. et al., 2013, Performance Analysis of Network I/O Workloads in Virtualized Data Centers. In *IEEE Transactions on Service Computing.*

Mietzner, et al., 2010. Combining Horizontal and Vertical Composition of Services. *In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications.*

Modica, G., et al., 2007. Dynamic Re-negotiations of SLA in Service Composition Scenarios. *In SEAA07, EuroMICRO conference of Software Engineering and Advance Applications.*

Oasis-open.org, 2007. Security Assertion Markup Language (SAML). *https://www.oasis-open.org/committees/download.php/27819/sstc-saml- tech-overview-2.0-cd-02.pdf.*

Oasis-open.org, 2012. eXtensible Access Control Language (XACML). *https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#C URRENT.*

Overton, C. 2002, On the Theory of Internet SLAs. *Journal of Computer Resource Measurement,* (106): 32-45.

ShaikhAli, A., Rana, O. F., Al-Ali, R., and Walker, D. W., 2003. Uddie: An Extended Registry for Web Services. *In Proceedings of IEEE Workshop on Applications and the Internet.* pp. 85-89.

Spillner, J., & Schill, A., 2009, Dynamic SLA Template Adjustments based on Service Property Bonitoring. In CLOUD'09. *IEEE International Conference on Cloud Computing.* pp. 183-189.

Tian, M. et al., 2004. Efficient Selection and Monitoring of QoS-aware with the WS-QoS Framework. *In WI'04, IEEE/WIC/ACM International Conference on Web Intelligence. pp.152-158.*

WS Architecture, 2002. http://www.w3.org/TR/ws-arch/.

Zhonghong, O., et al., 2012. Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2. *In Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing.*