Dual Validation Framework for Multi-Tenant SaaS Architecture

Zeeshan Pervez, Asad Masood Khattak, Sungyoung Lee*, Young-Koo Lee
Department of Computer Engineering
Kyung Hee University
South Korea
{zeeshan, asad.masood, sylee}@oslab.khu.ac.kr, yklee@khu.ac.kr

***Abstract*— Availability of enormous processing power and immense storage capacity drive Cloud computing. Information industry oracles like Google and Microsoft are delivering their next generation of products through Cloud. With the success of Amazon Cloud computing service everyone is laying on line to adopt it. Increased processing capabilities of enterprise servers do not mean that hosted services can handle infinite number of execution load. This massively escalating processing and storage capacity should be utilized effectively. Most of the current research is inclined towards its projected benefits and applications in real world. Being newly adopted, there is a need to set guide lines for Cloud computing paradigm. Myths about its capabilities could derail it, to avoid these misconceptions we have proposed a request validation framework which will help in limiting the processing load on services hosted in Cloud. The proposed framework works by validating the incoming request in its contextual semantics. The proposed framework helps in seamless service up gradation, which helps service provider in delivering quality of service by conforming their SLA.**

Keywords- Software as a Service (SaaS), Contextual Semantic.

I. INTRODUCTION

Over the time we have seen some dramatic changes in software delivery model: from stand alone applications to client server architecture and from distributed to service oriented architecture (SOA). All of these transformations were intended to make business process execution efficient and to provide ease of use. New software delivery model emerges due to the fact that either the earlier delivery model were not supporting the business needs or technological advancement have broken some barriers which were considered to be as inevitable in previous ones. Exponential increase in processing power of enterprise servers, adoption of virtualization and availability of high band width to the masses have given birth of new type of computing paradigm known as Cloud computing [1]. It encompasses Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Among these types, SaaS is a software delivery model, which provides access to business functionality remotely (usually over the internet) as a service [2]. SaaS addresses issues like service abstraction, reliability, data accessibility and interoperability.

* Corresponding Author. Email: sylee@oslab.khu.ac.kr

Leading companies in information technology industry are gradually moving their applications and related data over the internet and delivering them through SaaS [2]. Google has used SaaS platform to offer web applications for communication and collaboration [3]; gradually replacing resource exhaustive desktop applications. Similarly Microsoft is offering their development and database services though SaaS platform codename Microsoft Azure [4]. SaaS is preached by the companies like SalesForce.com, 3Tera, Microsoft, Zoho and Amazon, as a result of which business specific services can be consumed in ubiquitous environment.

II. MULTI-TENANT ARCHITECTURE

SaaS not only has altered the software delivery model, but it also has amended the software development methodologies. SaaS introduces the concept of Multi-Tenant, in which single instance of software is used by multiple users or even by different enterprises at same quantum of time.

SaaS can be categorized into four levels of maturity [5]. Services offered at Level-I, are Ad Hoc/Custom which dictates that separate instance of a service is deployed for each tenant. At second level (Level-II) services are Configurable; it is similar to the previous level, except that each service is configured according to the tenant business need; apart from that the core functionality provided by every service remains the same. Level-III is Configurable, Multi-Tenant-Efficient; previous two levels do not support multi tenant, but at this level services are designed to be accessed by multiple service consumer at same time, without compromising the data of other tenant. At Level-IV services are Scalable, Configurable, Multi-Tenant-Efficient. This level is same as previous level, however additionally support multi-tenancy and services are configurable according to business need. Level-IV focuses on quality of service, and Service Level Agreement (SLA) which is signed between service provider and service consumer.

All of the above mentioned levels fall under the category of SaaS, the fundamental difference is how these services are consumed by service subscribers. Undoubtedly Level-IV is most preferable for service provider because of its maintainability and scalability. Besides this Level-IV provides efficient utilization of Cloud resources and helps in reducing service provisioning cost.

978-1-4244-6949-9/10/$26.00 ©2010 IEEE

III. CLOUD COMPUTING AND ITS POPULARITY

Cloud computing includes Infrastructure as a Service, Platform as a Service and Software as a Service [6]. It has become a unique selling point for products which are in their development phases and even for the new versions of existing products. Recently Microsoft has released their office suite in Cloud computing environment [7]. Service consumer can subscribe to MS Office web service, thus leveraging their applications with SaaS. Service providers are defining strategies to deploy their commutation intensive applications in Cloud.

Search engines around the world receive enormous number of requests about Cloud computing. Figure 1 shows the result of Goolge Trend highlighting the popularity of Cloud computing in year 2009.

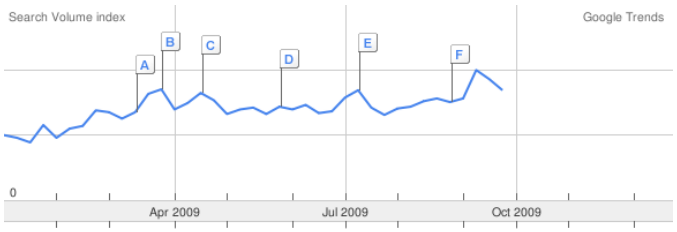

Figure 1. Google Trend for Cloud Computing

Introduction of new technology works as a remedy for existing problems; but it also brings in new threats and challenges. Same is the case with Cloud computing. CIO and CTO are rolling their sleeves up to adopt Cloud computing, and are offering their software over the wire as a service. Above and beyond service consumer are adopting Cloud computing to reduce infrastructure and service provisioning cost of their enterprise solutions.

A large number of research material has been published discussing Cloud computing and it potential usage in current business scenario. Ample effort has been made by Cloud computing evangelists to clearly draw a line between IaaS, PaaS and SaaS [1]. [8] and [9] presents their Cloud computing models which address different issues ranging from over all orchestra of Cloud computing, to security and performance issues in it. Most of the research is focused on what and what should not be delivered as a service.

Cloud computing provides access to massive processing and storage capacity as in the case of Amazon and Google. Availability of towering processing and storage capacity does not mean that Cloud can process each and every request. This is one of the myth about Cloud computing.

So there is need to design validation process, which can categorize request accordingly. Because of multi-tenancy property of Cloud computing existing validation process like XML Schema Definition (XSD) [10] cannot be applied efficiently. SaaS could be one of the stepping stone towards the adoption of Cloud computing. In order to make is more robust and to achieve quality of service, we have proposed validation framework targeting SaaS which we called Dual Validation Framework for Multi-Tenant SaaS (*D-Val*).

IV. DUAL VALIDATION (*D-Val*) FRAMEWORK FOR MULTI-TENANT SAAS ARCHITECTURE

D-Val is a dual validation framework in which incoming request is validated for it syntax as well as for contextual semantics. In enterprise systems web services are designed to handle complex business processes and result of each web service varies according to its context. In an enterprise a line manager has more privileges over data as compared to his subordinates, although same web service is used by both entities (line manager and subordinate) to generate some sort of report but the result will differ. In this scenario contextual semantics are very useful, in making decision on access privileges and input validations.

Figure 2 shows the validation process of *D-Val*. Dual validation process is achieved by means of XML and Semantic Rule Parser. Together these two parsers help in achieving SaaS optimization by efficiently filtering out the invalid request.

XML Parser works like an ordinary parser which validates the incoming request. The purpose of this validation is to reject any invalid request at the initial step of execution to minimize resource utilization. XML parser works by employing XSD. It verifies the incoming request for any of missing field and mismatched data type. Figure 3 shows XSD for a web service which query inventory database to generate stock status report for a particulate interval of time. This schema is useful in validating request parameters; to identify any missing and mismatched input parameters.

There are some limitations of XSD; it is very rigid; as describe previously behavior of web service varies from context to context. With XSD contextual validation is not possible. Having multiple XSD schema files for each diverse context will create confusion and will exponentially increase the efforts to maintain them.

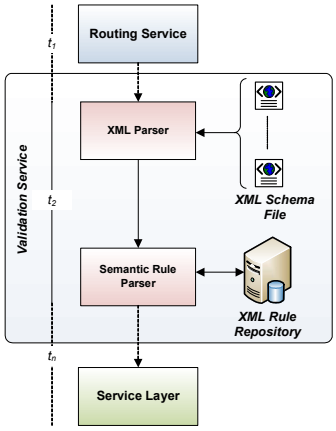

Figure 2. Dual Validation Process.

To address this issue we have introduced a Semantic Rule Parser. Semantic Rule Parser validates the incoming request contextually. Once request is validated by XML Parser, access privileges of the user are checked to avoid unnecessary error propagation by routing service. Semantic Rule Parser holds information about the access rights for individual users or roles. This information is encoded in the form of XML. Figure 4 shows the XML rules which are used to validate the incoming

request. It shows that for each user Semantic Rule Parser defines a node which helps in identifying the parameters required to execute the web service.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema
xmlns:xs="http://EnterpriseCloud.com/Schema">
<xs:element name="inventoryQuery">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="employeeName"
         type="xs:string"/>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="storeId"
               type="xs:string"/>
            <xs:element name="toDate"
               type="xs:string"/>
            <xs:element name="fromDate"
               type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 3. XML Schema of inventory request.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsr:element name="inverntoryQuery">
 <xsr:role>
  <xsr:element name="subordinate"
type="xs:string"/>
   <xsr:parameter>
    <xsr:element name="storeId" type="integer"
       isRequired="true"/>
    <xsr:element name="toDate" type="date"
       isRequired="true"/>
    <xsr:element name="fromDate" type="date"
       isRequired="true"/>
   </xsr:parameter>
  </xsr:element>
  <xsr:element name="manager" type="xs:string"/>
   <xsr:parameter>
    <xsr:element name="storeId" type="integer"
       maxValue = "" minValue = ""
       isRequired="false"/>
    <xsr:element name="toDate" type="date"
       maxValue = "" minValue = ""
       isRequired="true"/>
    <xsr:element name="fromDate" type="date"
       maxValue = "" minValue = ""
       isRequired="true"/>
    <xsr:element name="levelId" type="integer"
       maxValue = "" minValue = ""
       isRequired="false"/>
    <xsr:element name="productId" type="integer"
       maxValue = "" minValue = ""
       isRequired="false"/>
   </xsr:parameter>
  </xsr:element>
```

Figure 4. Semantic Rule of Inventory Request (Context Dependent).

The XML presented in Figure 4 also helps in deciding which parameter is required to execute the process but at a higher degree of level than XSD. This type of information cannot be modeled in XSD, for multiple users and roles. Referring to the example quoted earlier in which we have defined two roles for a web service, one is line manager and other is subordinate. Same service is used to entertain request for both type of role, regardless of the fact that both of them have different privileges over the data.

Functionality provided by the Semantic Rule Parser will help in web service maintainability, there is no need to alter service implementation if any of the role is changed; all what is required is to change the semantic rules. Most importantly these rules are defined in XML so no build process is required, which results in zero downtime during service up gradation.

One of the performance boosts we can gain is by storing these rules in database instead of file. A number of Database Management System (DBMS) vendors including Microsoft and Oracle support specialized data types of store XML contents. MS SQL Server 2008 supports declaring and storing XML content in XML data type. Performance can be improved by storing XML content in DBMS, as they provide replication and indexing services. These services will decrease the processing and load time of XML file regardless of their size. Furthermore these DBMS provides system stored procedure to process XML file [11] which could be used to reduce development and maintenance cost.

V. CONTRIBUTION OF *D-Val* TOWARDS SAAS ARCHITECTURE

D-Val is a validation framework for SaaS based Multi-Tenant applications. It leverages SaaS with data validation standard of XML along with contextual validation; and enhances its existing features to achieve QoS in Cloud computing orchestra. It works by introducing a rationale layer between realized services and routing service. With *D-Val* introduction we have gain advantages like

A. Request Filtering

In SaaS environment services are consumed by service subscribers through application programming interfaces like Facebook API [12] and recently released Google communication and collaboration product [13], in which service consumer sends the request and action is perform in Cloud. As services are consumed outside the federated environment of service provider, syntax and semantic of incoming requests may not confirms with realized service. With *D-Val* we can by far categorize incoming request into valid and invalid category, which helps in making decision on whether request should be executed or not.

B. Multi-Tenant Services

The core advantage of SaaS is Multi-Tenant nature of realized services; through which N number of service consumer can use single instance of hosted service (replicated virtually multiple times). With *D-Val* business rules are extracted from the service; by virtue of which only one implementation of service can be used to handle request form different tenants. So there is no need to separately host service which is specifically developed for a particular tenant.

C. *Quality of Service (QoS)*

QoS is one of the unique selling features of Cloud computing paradigm. Virtually replicating services on thousands of servers does not mean that we can handle infinite number of requests. *D-Val* helps in limiting the number of requests which could be executed on realized service. Instead of generating error response from service layer, which includes unnecessary processing power and memory consumption, *D-Val* verifies the request and decide whether it should be executed or not, consequently decreasing load on service layer and escalating quality of service.

D. *Green Computing*

Cloud computing introduces the concept of Green computing in which resources are efficiently utilized to minimize atmospheric pollution, by limiting CO_2 emission [6]. There is some cost affiliated with every request executed on the server; which could be calculated in terms of heat and CO_2 omitted. Through *D-Val* we have reduced the number of requests which could be execute on the server. Reduced amount of CO_2 emission and memory footprint results in more greener Cloud.

VI. Optimizing Service Performance with *D-Val*

The properties of Cloud computing are poles apart from other computing architectures. In Cloud everything is massively scale up from processing power to available storage capacity. In such an environment even optimization of one bit could result in terabytes.

Figure 5 shows how performance and response time of a service hosted in SaaS environment could be affected by adding validation layer between realized and routing services.

Figure 5(a) shows services in which validation and business rules are implemented implicitly. Service requests from variant type of users are handled by routing service. It acts like an entry point for the Cloud. On receiving request it delegates the request to service layer; irrespective of the fact that whether this request conforms to service implementation or not. Once request is delegated to hosted service, it starts the execution process which could be a simple or a complex operation in which interaction with other services is required. This increases the complexity as there are multiple services involved and error could be generated by any of the subsequent service. As a result the time spent by its calling service is wasted as process cannot be completed. Figure 5(a) shows the timeline in which request; valid or invalid will take $\sum(t_1 + t_2 + t_n)$ for its execution, which includes the time spend by the subsequent services to complete the process.

On the other hand Figure 5(b) shows the realization of SaaS architecture with validation service. In which incoming requests are handled by routing service. On receiving the request it delegates the request to validation service. Once request is received; Validation Service validates its input parameters with defined XSD schema and by the business rules represented in XML form. This helps in restricting invalid requests to be executed on realized services; which ultimately help in cutting down the execution, memory and routing cost (delegating request to subsequent service). Timeline in Figure 5(b) shows the difference in execution time of a valid and invalid request. Valid request takes $\sum(t_0+t_1+t_n)$ amount of time to complete the process where as invalid requests only spend $\sum(t_0+t_1+t_2)$ time. This shows reduced execution load on realized services, t_2 is far more less than t_n, ultimately increasing the overall response time of a service.

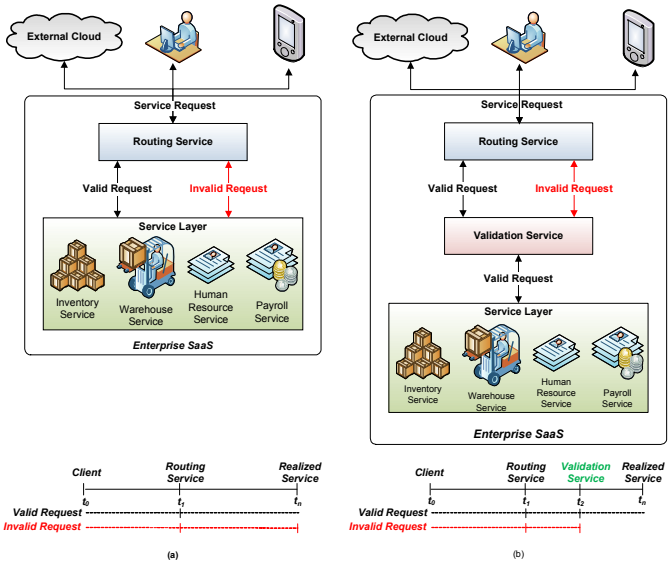

Figure 5. Performance gain achieved with D-VAL

Equation 1 shows the performance gain achieved in terms of time, by using *D-Val*. As described earlier that a valid request takes about $\sum(t_1 + t_2 + t_n)$ of the total time, in the case of invalid request it only take $\sum(t_0+t_1+t_2)$ time. By comparing time slot used by both type of request we can identify that total gain, which is a major improvement because we have salt time required by the subsequent services in case of invalid request.

$$\text{Total Time Gain} = -\left(\sum_{0}^{2} t - \sum_{2}^{n} t\right) \qquad (1)$$

VII. Conclusion

Escalating processing power and storage capacity does guarantee QoS, but there is always need of process optimization. Every service consumer wants to avail the benefits of Cloud computing orchestra, to lower service provisioning cost and to increase service performance. In SaaS single implementation of a service is consumed by multiple-tenant, thus increasing probability of invalid request. These invalid requests will degrade the service performance, as realized services will consume processing power on their execution.

With *D-Val* we have achieved twofold benefits. One is filtering out invalid request and handling them soon as they enter the system so that more complex services should no squander time on their execution. Secondly by extracting out business rules from the actual services we can make our services configurable to multi tenant environment. Only business rules will vary from tenant to tenant, rest of the implementation remains same for every service consumer.

Consequently *D-Val* facilitates development and maintenance activity and helps in achieving QoS and meeting service level agreement in SaaS architecture.

ACKNOWLEDGEMENT

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2009-(C1090-0902-0002)). This work was supported by the IT R&D program of MKE/KEIT. [2009-S-033-01, Development of SaaS Platform for S/W Service of Small and Medium sized Enterprises].

REFERENCES

[1] Sun, W., Zhang, K., Chen, S., Zhang, X., and Liang, H. 2007. Software as a Service: An Integration Perspective. In *Proceedings of the 5th international Conference on Service-Oriented Computing* (Vienna, Austria, September 17 - 20, 2007). B. J. Krämer, K. Lin, and P. Narasimhan, Eds. Lecture Notes In Computer Science, vol. 4749. Springer-Verlag, Berlin, Heidelberg, 558-569.

[2] Zhang, L. and Zhou, Q. 2009. CCOA: Cloud Computing Open Architecture. In *Proceedings of the 2009 IEEE international Conference on Web Services* (July 06 - 10, 2009). ICWS. IEEE Computer Society, Washington, DC, 607-616.

[3] Google Web Applications for Communication and Collaborations. http://www.google.com/apps

[4] Microsoft Windows Azure Platform. http://www.microsoft.com/windowsazure/

[5] Frederick Chong and Gianpaolo Carraro. April 2006. Building Distributed Applications - *Architecture Strategies for Catching the Long Tail.* http://msdn.microsoft.com/en-us/library/aa479069.aspx

[6] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.* 25, 6 (Jun. 2009), 599-616.

[7] Microsft News Center - *Microsoft Web Apps: Office Goes to the Web* http://www.microsoft.com/presspass/features/2009/sep09/09-17officewebapps.mspx

[8] Hudli, A. V., Shivaradhya, B., and Hudli, R. V. 2009. Level-4 SaaS applications for healthcare industry. In *Proceedings of the 2nd Bangalore Annual Compute Conference on 2nd Bangalore Annual Compute Conference* (Bangalore, India, January 09 - 10, 2009). COMPUTE '09. ACM, New York, NY, 1-4.

[9] Karabulut, Y. and Nassi, I. 2009. Secure Enterprise Services Consumption for SaaS Technology Platforms. In *Proceedings of the 2009 IEEE international Conference on Data Engineering* (March 29 - April 02, 2009). ICDE. IEEE Computer Society, Washington, DC, 1749-1756.

[10] W3C - XML Schema Validation. http://www.w3.org/XML/Schema

[11] Microsoft SQL Server Developer Center - *System Stored Procedure to Prepare XML Document*. http://msdn.microsoft.com/en-us/library/ms187367.aspx

[12] Facebook API. http://developers.facebook.com/

[13] Google Wave API. http://code.google.com/apis/wave/