A Framework for Cloud Embedded Web Services Utilized by Cloud Applications

Mohamed Wahib and Asim Munawar
Graduate School of Information
Science and Technology
Hokkaido University
Sapporo, Japan
{wahib,asim}@ist.hokudai.ac.jp

Masaharu Munetomo and Kiyoshi Akama
Information Systems Design Laboratory
Information Initiative Center
Hokkaido University
Sapporo, Japan
{munetomo,akama}@iic.hokudai.ac.jp

***Abstract*—Cloud computing is impacting the modern Internet computing and businesses in every aspect. One feature of clouds is the convenience of using the services offered by the cloud. Consequently, most cloud service providers use WS for users and developers to interface with the cloud. However, the current cloud WS are focused into core and fundamental modern computing functionalities. We anticipate as cloud developments tools mature and cloud applications become more popular, there will be an opportunity for designing and implementing applications/services to be embedded in the cloud for use by applications in the cloud. We propose a framework for WS deployment in the cloud to be usable by applications residing in the same cloud. The framework capitalizes on the cloud strong points to offer a higher value to the service consumer inside the cloud. The authoritative nature of clouds would enable more efficient models for WS publishing, indexing and description. Moreover, being hosted in the cloud, WS can build on the high scalability offered by the cloud with a much higher reliability. Finally, scheduling the instances using the WS in bundle with the WS instances could offer a LAN-like connectivity performance driving down the latency to the magnitude of lower microseconds. In this paper, we highlight the challenges and opportunities of cloud applications using cloud embedded Web services. We give a description of the different aspects by illustrating the different components, together with an end-to-end use case to show the applicability of the proposed system.**

***Keywords*-Web services; Cloud applications;**

I. Introduction

Cloud computing technologies are evolving in a high pace in analogy to how the Web1.0 and Web2.0 technologies changed the Internet from a primitive distributed file share and transfer platform to a complicated stack offering high level services. However, the current effort in cloud services is mainly focused on building high level abstractions to the cloud users enabling them to adopt the cloud technologies smoothly. The focus of this paper is on a business logic layer embedded in the cloud to offer an added semantic value to cloud applications. The new business logic layer builds on the WS standards as is the normal practice in cloud context. This paper proposes a framework for hosting CES (Cloud Embedded Services) in the cloud. To the best of the authors' knowledge, this has been no attempts for an open framework providing embedded services to applications in the cloud. The following is the motivation behind offering a service through CES rather than a traditional WS residing outside the cloud explained through an example. An example of a simple WS (we name it WS-Temp) receiving a city name and returning the temperature is used. For an application residing in the cloud, the first option is to use WS-Temp which is hosted in some other place in the Internet (or even on the same cloud but without the application and WS-Temp being aware). The cons in this scenario are the normal cons of WS accessed over the Internet in the current practice (latency, poor registry and description, uncertainty of reliability and availability). The second option would be hosting WS-Temp in the cloud and exposing it to applications running inside the cloud. A cloud application seeking to use WS-Temp in this case will have a unique opportunity for an efficient use of WS-Temp. This is tangible by examining the conventional WS, which in practice, suffer from problems in the reliability, scalability, high latency, indexing, registry and querying. This is logic as WS were designed to be hosted over a an open, decentralized and highly latent with low reliability platform; namely the Internet. This lead to loose WS technologies that relay heavily on the interpretation of the service provider. On the other side, with CES, as compared to conventional WS, a different set of challenges and opportunities come along from with the use in a cloud. This situation calls for harnessing the cloud strong points to elevate the mechanics of service computing into a new level. The main class of services that would be beneficial to the cloud applications and the cloud itself as an added value are services expected to be repeatedly used on a large scale by the cloud applications. For example, as the cloud applications involve a lot of data handling, a CES for data validation and checksum would be virtually used by data-oriented cloud applications.

The rest of the paper is as follows. The next section is a discussion of the significance of CESs and the opportunities and challenges of the concept. The following section discusses the architecture of the framework. Section IV illustrates a use case of the framework along with preliminary results. Finally section V concludes and adds insight to future work.

978-0-7695-4461-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERVICES.2011.56

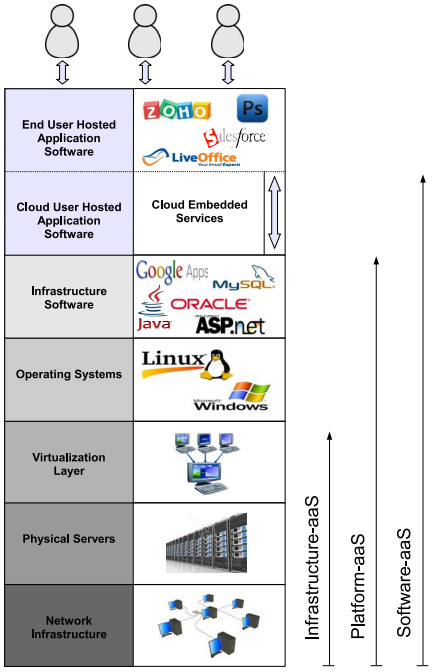

Figure 1. Service layers definition in clouds

II. Challenges and opportunities of Cloud Embedded Services

Before explaining the mechanics of CESs and how can CESs contribute in enriching the cloud environments, an insight of how services are utilized and incorporated in production clouds is essential. As shown in figure 1, WS are used mainly to interface the cloud offerings to the cloud users. These services cover all the spectrum of SaaS, PaaS and IaaS. These WS have the following characteristics: a) Limited in number, fast learning curve and fully documented. b) Do not need the traditional registering, querying and indexing. c) Accessible from outside the cloud. Also accessible from other WS interfaces in the same cloud (e.g. Amazon's EC2[1] can use simpleDB as part of the same solution).

CES, proposed in this paper, differ from the current WS used by clouds in the following: a) Accessible only from within the cloud, with a point-to-point connection (i.e. framework manager is aware of the binding). b) More CES would mean more functional services beneficial to cloud users, thus the framework allows for ease of adding CES by independent developers. c) Unconventional registering, indexing and querying of CES is essential. In short, CES are close from the functional aspect to the WS used over the Internet, while the current cloud WS are more into core and fundamental modern computing functionalities.

The proposed is to split the upper application layer into an end user layer at which cloud applications (see an example of the applications in the figure) and a lower layer of service entities providing semantic value to the upper layer.

Challenges and opportunities: Identifying the challenges and opportunities for the cloud provider, cloud user, end user is a essential to harness the potential power of CES over conventional WS to the cloud applications.

Opportunities :

1) Cloud provider: From the perspective of the cloud provider, offering a framework to host CES would offer the following benefits:

i) Give an added semantic value for the cloud user. As basic cloud WS are more oriented into core cloud functions, CES would give a higher level business logic layer to the cloud user. Some current cloud services can somehow be viewed in this regard (e.g. AWS MPI and MapReduce services), yet they still are not a high level semantic layer, they are still core cloud computing functions for the particular class of users they were designed to (High Performance Computing community in this example). An analogy here to the early days of the Internet is clear. Early Internet offered primitive data and files hosting+transfer for users (in correspondence of how clouds offer core computing services and infrastructure to cloud users). The Internet then evolved to a large stack of application and semantic layers over the initial primitive services.

ii) CES hosted in the cloud and cloud applications using CES would mean more computing done in the cloud and consequently more use of cloud resources. One scenario of CES designed to be offered by the cloud provider and another of offering a cloud WS interface to allow cloud users (a more accurate name would be producer cloud user) to add CES to the cloud. Both scenarios would mean more computing done in the clouds and more use of cloud applications to cloud resources.

2) Cloud user: The cloud user would be the most beneficial for CES, the following are the opportunities:

i) WS over the Internet suffered from an inherit problem of reliability and availability. Clouds, necessarily offering higher levels of reliability and availability, would refrain the cloud user from inconvenient failures and reducing the WS redundancy cost shifted to the cloud user.

ii) A main pillar in clouds, scalability, would also eliminate many problems for the WS provider in having to handle such problem. The evident model for scaling in the cloud would be highly beneficial to automate the scaling process of the WS in the highest efficient manner. Of course this advantage would be also the case for WS (not CES) hosted in clouds.

iii) The centralization of clouds compared to that of the Internet would be useful in designing a strict indexing and registry system for CES and consequently an efficient querying system. On the other hand, registry and indexing have always been a major problem in WS. To the extent that the major registry services have been shutdown and WS in

practice do not rely on registry. One other benefit would be using a strict description method for CES. The problem with description of WS is that the entire process is left to the service provider without any revision authority. For example WS-Temp can or cannot include in the description if the temperature forecasted is using a simulation or relying on actual data from a third party (another WS for example or live reports from meteorology monitoring in airports). To sum it up, the idealistic process of providing a WS over Internet would have less chances of failure over the clouds considering the centralization of clouds.

iv) The harder constraints of hosting a CES over the cloud would lead to a natural selection resulting in low redundancy CES(s) with high quality. For example, querying for a WS-Temp like service in WS available of the Internet would return a high number of hits without any quantitive measures to decide. Concern here is with the functional QoS attributes.

On the other hand, CES would have to go through the following to be available to cloud applications: a) be strictly adhering to the cloud rules in comprehensive description. b) Be of higher functional level than redundant CES to insure survival.

v) The previous point plus the authenticity of cloud users would enable for building a highly reliable reputation system to rank CES(s) depending on the functionality and not traditional non-functional QoS attributes in WS. vi) A major opportunity, the most relevant, is the ability to achieve LAN-like latency provided that CES(s) and the cloud applications utilizing them are allocated to the same data center. Latency have always been an issue in WS considering the millisecond range of Internet speed. However, scheduling in bundle (by associating CES instances and instances using them) and providing point-to-point connection between the CES instances and the instances using them would change the latency from the millisecond range to the lower microsecond range of high speed connections in data centers. Such a concept is already in use in AWS; instances of SimpleDB running over EC2 are claimed to have LAN-like latency. In other words, the cloud manager would consider both instances as a single solution. Note the difference here is that in AWS's case, the bundle is between two predefined core cloud services. While in the proposed, the bundle is between an application and a user defined entity.

3) End users: the benefit for the end user is passed along from the cloud user. Any added functional value, lower cost, less latency would be essentially tangible by the end user.

Challenges :

1- Cloud provider: Starting by the technical first:

i) The choice of the openness of the CES framework is a challenging issue. One side, the cloud provider could make it an closed framework with CES designed and hosted by the cloud provider just like current cloud WS. This would add a huge burden over the cloud provider and also be unfavorable by a significant class of cloud users skeptical about the off-the-shelf model the clouds are following just as it was in PC applications. On the other end of the spectrum, the cloud provider could have an open framework allowing for adding CES following a strict cloud protocol for CES quality and description. This would consequently bring along the defects of open systems. A model in between would be allowing developers to write CES, yet host them in the cloud after some form of cloud provider arbitration. A similar model is the one used by Apple's App Store which allows developers to develop Apps and upload them to the App store after arbitration from an Apple management entity. This challenge is a large topic to be discussed independently in a following publication. However, for the current paper, adding a new CES through the framework is unspecified whether it is directly by a cloud user, cloud provider or cloud provider on behalf of the cloud user.

ii) Security in hosting CES in the cloud is not more or less different from the normal security challenge of the cloud. The only additional challenge here is the point-to-point connection between instances of CES and instances using CES to cut down the latency. Such a communication level between two user defined entities inside a cloud can raise concern. This critical issue is outside the scope of this paper, yet is to be throughly studied independently.

iii) Scheduling instances over physical machines has an added challenge in this framework. Namely the bundled scheduling of CES instances with the instances using them. The complication is further aggravated when heavy use of a CES from cloud applications scattered would need some replication strategy.

iv) In a more complex scenario, cloud applications could need to utilize CES(s) in a workflow fashion. This would be challenging whether an existing message queuing service is used or a nouvelle module is built for that.

2- Cloud user: In comparison to using WS from outside the cloud, the challenges are as follows:

i) Obviously the security aspect comes again to the picture. The point-to-point connection with the CES needs to be securely designed.

ii) The legacy of using WS for around a decade accustomed the developers to the loose relation of their codes and the WS. In contradiction, cloud applications would be more integral with the CES. To achieve the best out of CES, applications' assignment and migration would be affected by the load and invocation patterns for the associated CES(s).

3- End user: Same as opportunities, and challenges for the cloud user will have an implicit effect on the service provided to the end user.

III. Architecture of CES framework

The architectural overview of CES framework is shown in figure 2. The core services are running in one or more VM instances. The core services are provided as Web services that can be accessed either directly using SOAP or by

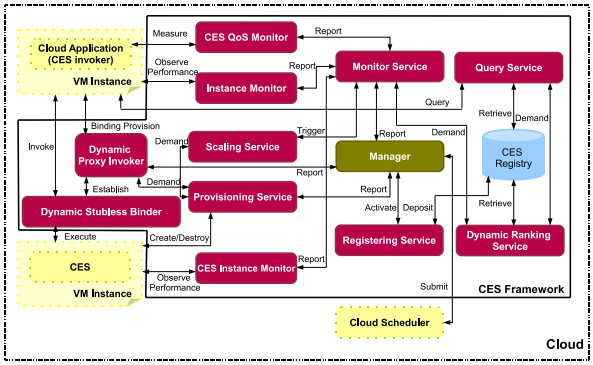

Figure 2. CES framework overview architecture

using the Client Library that provides a simple API. The framework in this paper was designed and implemented to run in an Eucalyptus[2] cloud. Eucalyptus Open Source is an open cloud software stack compatible with Amazon's AWS[1]. Two important points to clarify; a) The framework in this phase is designed for independent CESs. Due to space limitation, composite CESs are not addressed in this paper. b) Security in such a framework is complicated due to the inherit security challenges of clouds. The security aspect is covered in a subsequent publication while assuming a no access control policy in this paper. The modules of the framework function as follows:

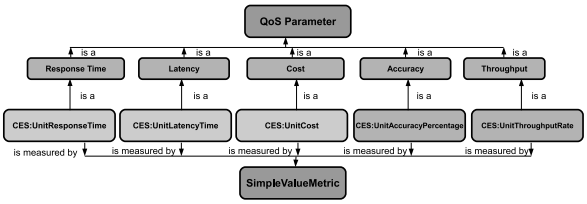

Figure 3. CES ontology for description of QoS parameters of CESs

CES registration: This section refers to the manner by which the framework handle newly registered CESs. The CESs registry (i.e. using UDDI and similar technologies in WS context is discussed later in the indexing and ranking section). CESs and associated metadata are stored in the Registry Database. The registry service accepts requests for adding CESs, process them according to cloud rules, register and index the CES, launch test instances hosting the new CES, and finally set the new CES to *active* status in the cloud. The authorized nature of clouds versus the openness of Internet gives an exceptional chance to improve the quality of the CES description compared to conventional WS. In this context, a registry WS was implemented (accessible from inside and outside the cloud) listening to requests. The request includes an EC2 image with the CES deployed inside it , OWL description file and an OWL extension file with the QoS thresholds and scaling triggers (i.e. a threshold level to be used for up scaling). An ontology extending onQos[3], an ontology developed using OWL for QoS description, advertising and query of Web services, was defined using OWL to describe the QoS parameters, see figure 3. The CES ontology is used for advertising and query of CESs, designed in order to ensure simplicity while maintaining flexibility and extendibility features. It is tied to the OWL-S ontology, which permits to connect a QoS description to the corresponding functional one. Following the classical approach for ontology definition, CES ontology is organized into three extensible complementary levels. The upper ontology defines the ontological language, which is the basic concepts to model CES QoS, such as the main properties and restrictions of QoS metrics. In this ontology, a QoS description of a CES is represented by a set of QoS metrics. For the QoS description of a service it is necessary to define a new entity of QoSMetric concept for each QoS parameter, which means to define: the measured parameter, the measurement scale, the measurement process, one or more measured values belonging to the measurement scale. The middle ontology is a specialization of the first one and is domain independent. The low ontology can contain domain-dependent specializations of the ontology in a specific domain. At this level some framework-specific concepts for QoS definition are introduced. The QoS parameters defined (e.g. *CES:UnitResponseTime* for query: maximum interval time within which the CES has to be executed; for advertising: interval time required to execute the CES invocation, It is expressed in seconds as float values.) are classified as *Simple Value Metric*, a specialization of generic QoS metric that permits to define queries adopting relation operators to be used on scale thresholding (such as better or equal, tightly less of a certain value, etc.).

Next the WS description is to be reviewed (possibly manually), then initially rated depending on description quality and accuracy. The following step would be generating an instance hosting the CES and run a throughput test for the CES in order to a) validation and integrity check b) test accuracy of description c) make a test run a test to ensure the CES is valid. Finally the CES would be indexed according to a rating and stated as *active*.

CES Monitoring: The monitoring service is split between monitoring the VM instances of the CESs and the applications invoking CESs from one side and monitoring the CESs non-functional QoS by regularly measuring the current QoS values from the other side. A central monitoring service orchestrates the entire monitoring. The instance system monitoring is based on GroundWork Monitor[4], which is an open source cloud monitoring tool supporting the Eucalyptus cloud stack. For the QoS parameter monitoring, conventional methods (i.e. with access to the CES implementation) are not possible as the CES comes in a black box fashion (i.e. embedded in an EC2 instance). Therefore, the method proposed in [5], a three-phase WS independent invocation based QoS

measurement, is used for QoS periodic evaluation.

Dynamic proxy invocation: A highly challenging aspect of the framework is the CES invocation scheme. The conventional precompiled client-stub tightly-coupled invocation scheme of WS (whether it uses SOAP or REST) is not adequate to be used in a highly dynamic system requiring the decoupling of the CESs and applications invoking them. Applying auto scaling and dynamic binding (i.e. both CES instances and the application instances invoking them are dynamically created) would require an invocation scheme that is a) dynamic b) message-driven and c) asynchronous. DAIOS[6] is web invocation framework which solves this problem through a message-driven invocation system which translates invocation messages to SOAP/REST invocations dynamically. The scheme proposed in DAIOS solves partially the issues in CES invocation requirements; namely dynamic asynchronous decoupled invocation. The limitation of adopting DAIOS in the CES framework is the scaling factor. In the CES framework, the message translation can be a bottleneck for high orders of magnitude of concurrent invocations. Possible solutions could be a) introducing batch-invocation; invocations are processed in batches if a high magnitude of invocations occur simultaneously. b) adopting a distributed invocation scheme where each CES instance would process it's own message in the front end (without the CES awareness). In this paper, the scaling factor is left out to be studied independently and the DAIOS invocation framework extension of Apache Axis 2 was used as it is.

Scaling service : An integral part of the framework is the scaling of CECs in the cloud. Scaling the CESs is as follows:

i) The monitoring service triggers an alert if any of the QoS parameters value fall outside the scaling threshold defined by the CES developer in the OWL extension provided with in the registration phase.

ii)v) As an EC2 instance provisioning may require a long time (in case of a large image), the triggering alert from the monitor service is given well ahead to have the new instances up and ready for new invocations and avoid any latencies.

iii) The manager signals scaling service which demands the provisioning service to provision a new CES EC2 instance(s). The number of instances provisioned depends on the regression of the QoS parameters degradation (i.e. the faster the drop in throughput was, the more number of newer instances required).

iv) The manager assigns the new incoming invocations for the instances with the highest QoS utility function value (the equally-weighed sum of all QoS parameters). Further improvements by using a load balancing mechanism and replica scheme is certainly to be considered in the future work.

v) CES instances without any invocations are terminated.

Indexing and ranking service : First and foremost, service registries such as UDDI did not grow as expected. Even the major public UDDI registers were shut down several years ago. This setback was due to registries assuming voluntary registration by service providers (registry is passive, as opposed to actively crawling the Web looking for WSDL definitions of services). One more reason, Registries did not provide any value-added service, such as checking the quality of the registered services. As a result, service registries are often missing in service-centric systems (i.e., no publish and find primitives). This leads to point-to-point solutions, where service endpoints are exchanged at design time (e.g., using e-mail) and service consumers statically bind to them. Several solutions in the literature tackle this problem. [7] identify the limitations in registration and assume a new approach to SOA not relying on registry.

The situation here is obviously different for CES. The authoritive nature of clouds allows for an affirmative action from the CES developer to load his CES into the cloud. An important design aspect in the proposed framework is to utilize an efficient description model and past usage for pre-registration validation. By that we target to allow the service developers (and not just the service consumers as in convention) to make use of other registered services. Back to the example of WS-Temp, a CES developer seeking to add a new CES named WS-Temperature that provides some meteorological data would first query two things; a)if any CES having similar functionality is active in the cloud, notice here that the strict description system should offer comprehensive information about WS-Temp. b) for comparison's sake, the developer can retrieve logs of WS-Temp performance both from the cloud side and the CES side. Also the rank of WS-Temp in the cloud can be retrieved (CES ranking system will be mentioned later). The indexing service in this framework builds on a UDDI registry with the following added functions: While a comprehensive semantic query system would be ideal, for the time being the query not only returns the keyword-based matching CES. It also returns performance reports of the CES. The performance reports include QoS parameters stack graph, normalized number of hits and reputation ranking.

As for the ranking, in traditional WS, it is occasionally mentioned in literature in the process of selecting Web services from a large number of potential services in Web service composition. Several ranking techniques were utilized, most notably based on the semantic annotations of a service's inputs, outputs and functionality, as well as preconditions and effects, if available. We refer the reader to [8] for more details. Hosting CES in the cloud with the applications using them would create two potential opportunities in ranking CES; a) ranking systems in literature are working on-demand bases, so the ranking is calculated upon user request. In CES case, having the CES running inside the cloud would allow for a dynamically updated ranking system based on the usage pattern. b) having direct

Test Run		Average Response Time Per Invocation (Sec)	Total Number VM Instances Used	Average Time (Sec) ± Standard Deviation			
Number CESs	Average Number Ivocations / CES			CES VM Image loading	Binding Overhead (From Client Invocation to Actual Invocation)	Latency Overhead	Auto Scaling Overhead
50	100	43.4	101	27 ± 0.503	2.120 ± 0.087	0.0075 ± 0.00048	4.054 ± 0.593
100	90	42.6	141	29 ± 0.622	2.602 ± 0.094	0.0073 ± 0.00050	4.008 ± 0.873
150	80	40.2	182	29 ± 0.671	3.084 ± 0.069	0.0065 ± 0.00031	3.840 ± 0.605
200	70	37.7	221	26 ± 0.483	3.830 ± 0.039	0.0084 ± 0.00067	3.762 ± 0.774
250	60	36.5	263	31 ± 0.622	4.426 ± 0.104	0.0059 ± 0.00073	3.467 ± 0.680
300	50	34.4	312	30 ± 0.742	5.126 ± 0.083	0.0091 ± 0.00058	3.472 ± 0.792
350	40	32.7	359	35 ± 0.512	5.016 ± 0.073	0.0062 ± 0.00039	3.420 ± 0.530
400	30	30.4	407	28 ± 0.490	4.684 ± 0.041	0.0082 ± 0.00082	3.229 ± 0.980
450	20	29.8	452	30 ± 0.464	4.076 ± 0.062	0.0078 ± 0.00059	3.217 ± 0.429
500	10	27.1	502	32 ± 0.384	3.273 ± 0.083	0.0087 ± 0.00070	3.202 ± 0.839

Table I
CES IMAGES AND TIME PORTIONS FOR DIFFERENT FRAMEWORK MODULES.

access to the entire cloud, new factors for ranking can be taken into consideration. Mainly the factors based on the performance of the CES independently from underlying structure. c) ranking information collected entirely inside the cloud would give more reliability to the ranking system. The ranking system currently implemented is based on the QoS evaluation system mentioned earlier. Another factor is the normalized number of invocations. Lastly, a CES survival rate (i.e. the percentage of CES life time it was under invocation). The equally-weighed sum is used to dynamically rank the CESs and not on-demand base.

Scheduling the CES instances : scheduling in the framework is left entirely to the cloud manager. One important point here is to assure that the CES instances and the instances of the applications invoking them are allocated to the same geographic location (assuming multiple sites to the cloud). This can benefit from the expected LAN-like communication speeds inside a geographic location of a cloud. This would drive down the latency, which is a main weak point of WS over the Internet. An important note here, the entire framework does not affect the lower cloud middleware layer except for scheduling as explained here. To benefit from high communication speed and drive down the latency, some level of control over the scheduling process by the CES framework would be assumed. For the current version, the test bed used is located in one geographic location, so LAN-like latency is achieved without need to intervene as will be shown in next section. Future improvements will handle the scheduling over multiple geographic locations.

Manager : is a service orchestrating the interaction between the services in the framework.

IV. USE CASE AND PRELIMINARY RESULTS

Experiments for registering CESs and then generating invocations were conducted. The experiments were run on a 18 node cluster with 2 x AMD Opteron 2.6 GHz Dual Core 64 bit processors and 2GB RAM for each node. Eucalyptus 2.0 is used as a cloud stack, the communication

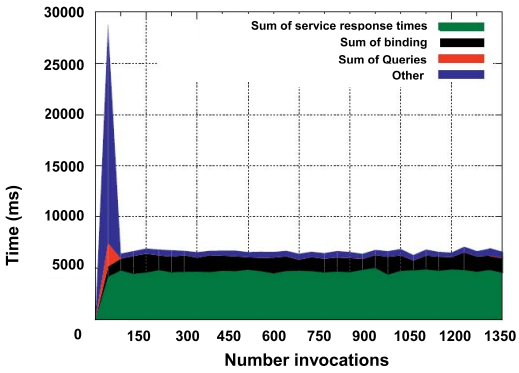

Figure 4. End-to-end performance of the framework

used is a 1GB Ethernet. We have created different sets of test CESs and QoS configurations (with varying response times) using the Web service generation tool GENESIS[9]. A Gaussian distribution is used for specifying the response time of all CESs. The number of invocations is increased while decreasing the number of CESs to show the effect of scaling on the overall performance (i.e. an increasing number of replicate instances required for CESs with high invocation rate.). Table I shows the settings of the test runs with different overheads. Two important points to note. First, the total number of images invoked increase significantly with the increase of number of invocations despite the lower number of active CESs. Second, the binding overhead tend to increase significantly with the increase number of invocations handled by framework (in the test run of 300 CESs and 50 invocations per each). In figure 4, we show the average process duration study based on 300 active CES and average of 50 concurrent CES running one images under the scaling threshold. Each CES continuously executes the incoming requests until finished. The incoming requests use

a Gaussian distribution with short intervals to emphasize on the scaling factor.

V. Conclusion

This paper proposed an open framework running in the cloud to host WS consumed by applications residing in the cloud. The main motive is to offer a highly dynamic and efficient environment for delivering high level semantic value to cloud applications. The proposed framework, CES framework, meets several challenges considering that WS we originally designed to a highly latent non-authoritive environment (i.e. the Internet). The challenges of the invocation scheme, QoS parameter description, indexing and monitoring were investigated. Opportunities as well appear when capitalizing on the cloud strong points for the framework. The high availability, ability to auto scale and driving the latency down to LAN scale can show a high improvement in CESs compared to conventional WS. A test case showing, using the Eucalyptus cloud stack, the applicability of the framework is illustrated to give a complete view of all aspects.

As for future work, many points can be improved to better adapt CESs in the cloud domain. An ontology based query scheme which takes into consideration the QoS parameters and history of the CESs in the framework is one point. Another point is to handle the scenario of multiple geographical-sites cloud in the scheduling process. An important point also would be improving the message-based invocation scheme considering the scaling factor.

References

[1] [Online]. Available: http://aws.amazon.com/

[2] [Online]. Available: http://open.eucalyptus.com/

[3] E. Giallonardo and E. Zimeo, "More semantics in qos matching," in *IEEE International Conference on Service-Oriented Computing and Applications*, 2007.

[4] [Online]. Available: http://www.groundworkopensource.com/

[5] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping performance and dependability attributes of web services," in *IEEE International Conference on Web Services (ICWS06*, 2006, pp. 205–212.

[6] P. Leitner, F. Rosenberg, and S. Dustdar, "Daios: Efficient dynamic web service invocation," *IEEE Internet Computing*, vol. 13, pp. 72–80, 2009.

[7] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-end support for qos-aware service selection, binding, and mediation in vresco," *IEEE Trans. Serv. Comput.*, vol. 3, July 2010.

[8] R. Wang, S. Ganjoo, J. A. Miller, and E. T. Kraemer, "Ranking-based suggestion algorithms for semantic web service composition," in *Proceedings of the 2010 6th World Congress on Services*, ser. SERVICES '10, 2010.

[9] L. Juszczyk, H.-L. Truong, and S. Dustdar, "Genesis - a framework for automatic generation and steering of testbeds of complexweb services," *Engineering of Complex Computer Systems, IEEE International Conference on*, 2008.