Model-Driven Architecture for SLA Orchestration in Cloud Services Based Systems

Assel Akzhalova, Akylbek Zhumabayev and Ulan Dossumov
[1]*Department of Electrical Engineering and Computer Science, Kazakh-British Technical University, Tole bi, Almaty, Kazakhstan*
a.akzhalova@kbtu.kz, akylbekz@gmail.com, ulan.dossumov@gmail.com

Keywords: Model-Driven, QoS, SLA, cloud, cost, protocol.

Abstract: The aim of MDA is to facilitate an entirely component-based system generation from models which are presumably small and manageable. It enables the system developing process be scalable meeting fast growing e-business demands. The system definition can be mapped to many possible platform infrastructures including those that are built on cloud services. Basing on MDA concept cloud services can be introduced by different providers offering and loaning available physical resources for a limited requested time by predefined contract containing service specifications. The service specifications and service provider's obligations are the elements of Service Level Agreement (SLA) that is a contract over the quality of service (QoS) and the violation of QoS may be a part of the contract between parties described as penalty policy. This paper proposes a system architecture of cloud computing infrastructure with dynamic QoS and SLA included that enables fair cloud resources facilitation.

1 INTRODUCTION

An efficient mechanism of interactions between clouds provides a way to enhance the capabilities of one of them. Such inter-cloud resource sharing requires dynamic resource allocation. Using Service Level Agreement(SLA) is vital in such dynamic environment. The cloud consumers expect high quality services delivered by certain cloud computing providers since they pay for them.

An example of use-case dynamic cloud service request and approval by the merging of providers' resources according to SLA and and Qos can be formulated as follows. We assume that there is available information about providers with the same cloud services. The user' requests can be handled by one provider that can request additional resources from the given pool of providers which is possible only according to predefined SLA. In other words, we can consider user as the provider requesting for resources in case if he needs some. Therefore, providers can create QoS documents to describe their minimum expectations from their services. The violation of QoS causes some form of penalty for participants. As a result, the provider reconfigures SLA object if there occurs any need to change current configuration of its cloud. Hence, a provider can share his available resources with others who needs more resources due to overload. When users request the service from the provider they have to agree on certain parameters of the SLA before they can continue to use the cloud. Upon successful agreement the user is permitted to reserve a certain amount of resource that is described in the SLA. Obviously, providers have to keep track of resource usage of every user. Afterwards, involved providers charge users based on their usage history and pricing plans. Figure 1 below shows described scenario as particular use-case.

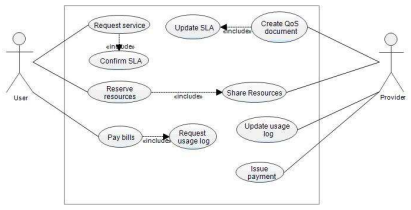

Figure 1: Use-case of user and provider interaction modeling of SLA and QoS-aware cloud service system.

Figure 1 demonstrates run-time system of the SLA and QoS based framework that typically may happen in case of shareable resources by cloud providers already signed SLA and cannot change fixed parameters and, therefore, do not take into account dynamic cloud service allocation.

Akzhalova A., Zhumabayev A. and Dossumov U.
Model-Driven Architecture for SLA Orchestration in Cloud Services Based Systems.
DOI: 10.5220/0005885500560064
In *Proceedings of the Fifth International Symposium on Business Modeling and Software Design (BMSD 2015)*, pages 56-64
ISBN: 978-989-758-111-3

For instance, (Keller and Ludwig, 2003) describes the Web Service Level Agreement (WSLA) framework that define and monitor SLAs for Web Services using WSLA specification language. The WSLA language is based on XML and it is defined as an XML schema. According to WSLA, Service provider signs up a contract that is the WSLA document to perform a service according to agreed guarantees for service level parameters such as response time and throughput, and measures that have to be taken in case of violation and failure to meet the asserted service guarantees (Figure 2).

```
<!– Global WSLA structure –>

<xsd:complexType name="WSLAType">
 <xsd:sequence>
  <xsd:element ref="wsla:Parties"/>
  <xsd:element name="ServiceDefinition" type="wsla:ServiceDefinitionType"
                        maxOccurs="unbounded"/>
  <xsd:element name="Obligations" type="wsla:ObligationsType"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="SLA" type="wsla:WSLAType">

</xsd:element>
```

Figure 2: WSLA structure.

The typical template of WSLA consists of three main parts: Parties, Service Definitions and Obligations. Parties means the parties involved in the management of the Web Service. Service Definitions contains an information about the services the WSLA is applied to such as service actions and service parameters and metrics. It also describes measurement methods of a service's metrics. Obligations specify the service level that is guaranteed with respect to the SLA parameters. A service provider can specify price for each providing service. However, the price must be defined in advanced and it is fixed regardless of current loading or provider's (consumer's) preferences.

Thus, one of the most important problems that providers of cloud services have to tackle is the fair method to charge customers for the real cloud utilization. However, the frequent resource allocation upon new arrivals of tasks make it a challenging problem to construct the system containing a combination of available resources along all the SLA parameters that would cater user requirements.

Ideally, once when current service becomes expensive to use by current customer, the system should automatically binds cheaper but faster service in order to meet SLAs and QoS. QoS and SLA monitoring are able to retrieve important parameters from the available running services which can be used in future for better adaptation. This approach can be an attractive solution in more complex case when cloud service providers might organize temporal agreements for cloud merging.

Thus, an efficient solution of service-oriented cloud computing should consider dynamic configuration of SLA that varies across all stakeholders of the cloud (Groleat and Pouyllau, 2011). The providers should understand a potential risk of customer dissatisfaction caused by delivery of services of low quality. Business operations force QoS requirements to be dynamic and adjustable to the operational environment.

In this paper we present a meta-model of the cloud services based system applying MDA. The MDA based framework can be considered as one that finds best cost for each cloud provider at run-time in order to provide maximum profit to keep desirable performance and QoS parameters. This architecture can be scaled up by allowing the deployment of multiple service instances running on different cloud servers. We analyze interaction between the system performance, QoS and SLA measures and introduce monitoring engine as an additional instrumentation of SLA observers that are generated from the initial meta-model.

The choice of appropriate cloud service resources is defined by policies that show QoS characteristics and SLA parameters. It is essential to design and implement admission control mechanism that will be able to conduct best cloud service allocation in order to introduce cloud service composition framework supporting QoS.

According to the generated SLA observers and QoS we introduce SLA based orchestration for flexible and fair cloud resources facilitation and meantime providing best deal for cloud service providers. Section 2 introduce MDE approach for SLA and QoS-aware system architecture. Section 3 describes SLA observers and generation cloud merge protocol for the proposed architecture. Section 4 demonstrates Case study. Section 5 compares different techniques that base on policy-aware service composition. Section 6 summarizes contribution and results.

2 MODEL-DRIVEN ENGINEERING FOR SLA AND QOS-AWARE SYSTEM ARCHITECTURE

Model-Driven approach provides interoperability and reusability of SOA systems by settling and manipulating a set of interoperable metamodels at different layers of abstraction. The role metamodel is mainly to enable future automatic transformations between models which makes the software development more efficient. The specification and design of ser-

vices within a service-oriented architecture proposed by OMG standard introduces Model Driven Architecture (MDA) to model SOA. MDA for SOA separates the logical representation of a service from its possible implementation on various platforms through auxiliary transformation the platform independent model to a platform specific model by maintaining mapping functions which are collection of rules and algorithms. In SOA prospective, these rules can be regarded as policies for a design-time and run-time service composition via model transformation (see for example, (Gönczy et al., 2009), (Bocchi et al., 2008), (Cortellessa et al., 2007)).

The life cycle of policy enforcement on these levels includes policy description and repository, adaptation and monitoring. The policy repository stores format or policy expression and can be formalized by simple using constraint language. Adaptation is conducted under service control engine. Service control engine selects a service provider from a number of service providers having their service details, in particular: service quality attributes and SLA.

This work presents a model-driven framework for policy generation to select the service provider as matching technique that comprises of judging from negotiation of service functionality and cost. We include to the system architecture monitoring component that collects metrics or statistics to support the process of checking for policy compliance. After observing the adaptation for an initial policy definition, the policy may be modified to better align with the specific objectives. In particular, we suggest that the relationship between possible reconfigurations and QoS constraints should be incorporated into the design of a SOA: the additional complexity providing significant benefit at runtime through automated policy generation. Our focus will be on meeting QoS constraints (performance and reliability) for an overall architecture, what we consider as Service Level Agreements (SLAs). Figure 3 shows the system architecture of handling requests by several cloud providers using proxies and controllers.

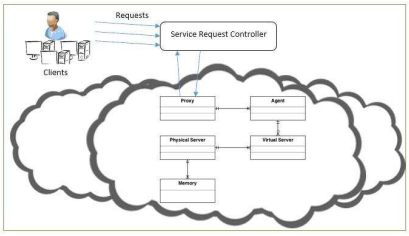

Figure 3: An architecture of the cloud system handling requests by multiple providers.

In Figure 3 there are several parties and components: Users, Service Request Controller, Physical Machines, Virtual Machines, Agents and Proxies.

Users/Clients Users are involved in the system since they send service requests to the clouds for processing.

Service Request Controller is an interface between the clouds and users/clients. It is responsible for resource management and allocation. Governed by SLAs, it manages the interaction of the clouds in order to fulfil QoS requirements set by users and other providers.

Physical Machines . As every provider has physical machines that includes memory and storage physical machines are responsible for consuming, processing and monitoring requests on physical level.

Virtual Machines can run on a single physical machine. They act as a logical layer in the operational hierarchy. VMs are responsible for physical resource allocation, memory usage, processing and monitoring requests on application level.

Agents are responsible for storing information on their providers, such as: current bandwidth, memory, storage capacity, CPU load and current number of requests.

Proxies We assume that each provider has its own set of proxies that receive requests particular for its provider. These proxy servers eventually tell controller about their throughput, number of requests and server load such that controller can reallocate resource among different cloud service providers based on SLA.

The scenario of processing user's request works as follows:

First, user's request for a particular service *S* from Provider *A* goes to Service Request Controller. Service Request Controller redirects the request to the provider's Proxy. Proxy of the specified provider sends the request to physical servers that can process the request. Service Request Controller redirects the request to the provider only if there is a confirmation from provider that it is able to proceed. Otherwise, if original provider cannot handle the request by itself, it requests extra resources from other predefined providers based on the SLA between them. Meantime, an information about status of providers are monitored by Agents that provide current loading of virtual machines in order to fulfil SLA parameters.

In our work we offer MDE for cloud based system that identifies meta-model and relationships between the following components: Cloud based system, Controller, and Monitoring. The metamodel includes:

- a QoS constraints defined by UML for QoS profile;
- model transformations that: 1. generates SLA objects taking QoS specifications of how an architecture should evolve in the face of QoS constraint violations for the adaptation engine; 2. generates self-organized pattern for SLA protocol that Agent constructs automatically in order to quickly to changes the service binding.

As it can be seen from Figure 4 the dynamic service composition can be implemented by applying policies which are based on QoS requirements and SLAs. The service handling can be considered as model transformations MT): MT1, MT2, MT3 (Figure 4 which makes reconfiguration of the architecture.

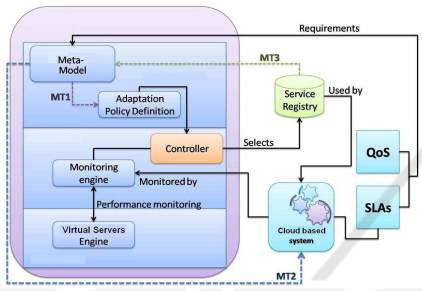

Figure 4: Model transformations for the Cloud based infrastructure.

The model transformation MT1 is able to automatically frame the appropriate cloud provider which is realized by Open Shortest Path First (OSPF) algorithm to find the clouds that have highest throughput and is available to share resources at the time of particular request, over which our dedicated Agent can determine reconfiguration strategies (choices over variant points) as a function of environmental changes. This resulting policy table is then combined with a mapping from choices to actual actions on the implemented system, to provide a runtime adaptation engine.

We employ model transformation MT2 in order to extract application metadata from the design time metamodel, with the purpose of understanding how the system is configured at runtime and, consequently, what needs to be monitored:

- Deployment data for individual services (location, interfaces, etc);
- The initial architectural configuration of services (what usage connections exist between services);
- The basic properties that are necessary to compute values of the QoS characteristics used in the model: memory, number of requests, throughput, response time and storage capacity.

A third model transformation MT3 is then used over this monitoring information to change the information associated with individual services in the repository model. Policy generation, as it will be discussed in the next section, is modeled at design-time as a possible transformation that an architecture model can undergo, representing possible reconfigurations of service composition based on self-organized pattern. Therefore, service selection allows us to consider the reconfiguration of an architecture as a *transformation* from one SOA for Cloud based model instance to another.

3 AUTOMATIC SERVICE COMPOSITION BASED ON SLA AND QOS

The metamodel is equipped with QoS characteristics and SLA that computes the overall cost as a function of time of architectural configurations that has to be minimized.

QoS parameters are defined at runtime due to the dynamic negotiation processes upon new service requests. For instance, required amount of memory, bandwidth, storage capacity, response time, throughput are QoS parameters that are embedded to SLA which calculates the cost. At the stage of MT1 the new SLA object *newSLAObject* is created and it consists of the following fields:

- memory
- number of requests
- throughput
- response time
- storage capacity

The cost of handling the service we calculate taking into account switching price between k providers, m VM that able to process request by allocating memory and storage as well as time penalty if the request is processed overdue and price of data transfer:

$$cost = N \cdot \left\{ \sum_{i=1}^{m} (CostVM_i + CostMem_i \cdot 3600 + \right.$$
$$\left. + StorCost_i \cdot 3600) \right\} + TranferCost +$$
$$+ \sum_{i=1}^{k} SwitchCost_i - \sum_{i=1}^{m} PenaltyCost_i$$

where CostVM$_i$ - cost of using a virtual machine i triggered by the incoming request (*perhour*), CostMem is the cost of using memory belonging to working VM (*persec*), StorCost is the cost of using a physical storage (*persec*), TransferCost is the cost of data

transfer per hour (*perGb*), SwitchCost is the cost of requesting resources from an additional cloud, PenaltyCost is the time penalty (*persec*), *N* is a number of requests, *m* is a number of virtual machines triggered by all user requests during an hour, *k* is a number of additional clouds (providers) triggered to process all requests during an hour.

SLA negotiation is based on the constraint satisfaction. For instance, preferred price set by a requesting provider for a particular service should be either declined or accepted by an offering provider. However, our main goal is to maximize profit for all providers by sharing their idle resources. As a result, overloaded cloud service providers will still be able to respond to user requests by leveraging available resources from other providers having low resource consumption. At the same time, it will be beneficial for providers to lend their resources instead of keeping them idle for uncertain period. Thus, the values of the variables in 3 are negotiated during the service discovery and binding. The process of dynamic allocation of resources to meet SLA is autonomously handled based on the following Policy which is implemented by Service Request Controller:

1. A new request is received
2. SLA negotiation process is starting
3. Estimating QoS
4. If estimated QoS violated desirable QoS then Service Request Controller analyzes the number of extra resources and finds a third-party provider to satisfy QoS requirements by forwarding requests to negotiated resources issued by the third party.

For instance, Service Request Controller handles SLA negotiation process and estimates time duration of processing a request by cooperation with Agents and Proxy of current provider. The set of proxies that implement Proxy engine informs the Agent about current workload of VMs including available memory, CPU and number of requests.

MT2 is considered as Protocol that facilitates the negotiation process between participating cloud computing providers. Specifically, Protocol enable automatic procedure of service discovery and binding by the middleware between providers based on their functional and non-functional requirements.

Cloud providers either reserve or lend resources by constantly adjusting prices based on individual strategies and it might be represented in a form of competitive game, in particular, so-called winners in the minority. In other words, the worst 'player' in the majority will have to change its protocol parameters. Krothapalli and Deshmukh describe this scheme where cost is considered together with a process completion time (Krothapalli and Deshmukh, 1999). The amount of cost depends on the due date and the processing time. A customer requests bids from the providers. After some period the customer may refresh the request again to eventually select a certain bid.

In this work we introduce model transformation MT2 that allows automating the configuration of the cloud providers negotiation and generates Protocol which is designed to provide the newly connected clouds having configuration parameters required by QoS and specified in SLAobject. Basing on SLAobject the new cloud should be able to communicate with another cloud automatically as shown in Figure 5.

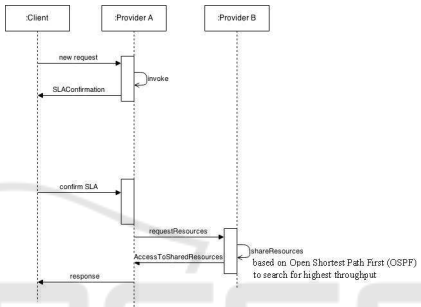

Figure 5: Protocol of binding services.

Protocol is derived from Open Shortest Path First (OSPF) to search for cloud provider with the highest throughput. An original form of OSPF calculates the shortest path to all known destinations connected to the network (Le Sauze et al., 2010). In our metamodel we employ this algorithm to share resources according to the highest throughput which is taken into account in MT1 to calculate cost. OSPF protocol has the following states: Down state, Init state, Two-way state, Exchange start, Exchange, Loading, Full adjacency.

In our case, we refer Down state to the state when particular cloud is inactive. The inactive cloud does not exchange any messages with others and waits for the next state. Initial state we use for the state when particular cloud sends "Hello" packet to the Service Request Controller in order to learn about all active clouds participating in the network. In Two-way state state the cloud that has sent packet "Hello" receives the packet back which means the connection has been established between the cloud and the Service Request Controller. Afterwards, the cloud can change the state to Exchange state in which it actually starts sharing its own resources. Loading state is the state when the cloud starts being loaded up. When the

cloud is involved actively in the sharing process then its state becomes Full Adjacency in which it does not need to send initial packets in order to introduce itself to the available pool of cloud providers.

Therefore, we can demonstrate Protocol described above as follows in Figure 6, Figure 7 and Figure 8. For instance, Figure 6 considers step when new cloud joins the composition it should send ANNOUNCE request to introduce itself to the group members. Figure 6 considers step ASKRESOURCES to request needed resources and request ASKRESOURCES is immediately followed by the request SHARERESOURCES in order to ensure that the requesting cloud is still operating shown in Figure 8.

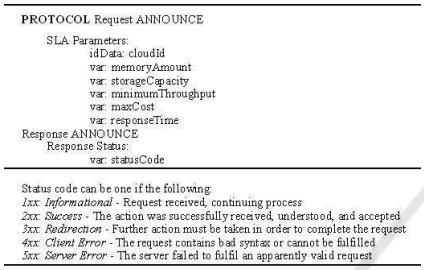

Figure 6: Protocol: Step ANNOUNCE.

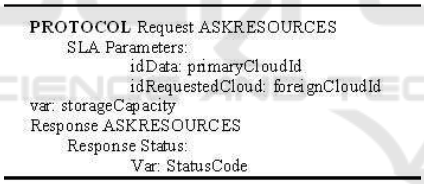

Figure 7: Protocol: Step ASKRESOURCES.

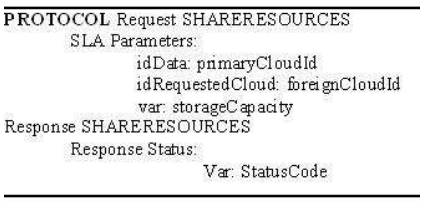

Figure 8: Protocol: Step SHARERESOURCES .

We consider QoS constraints as it was presented in paper (Akzhalova and Poernomo, 2010). After cost model has been defined and QoS requirements are determined and included into SLAobject the model transformation MT3 automatically changes the global configuration of the system to adjust it to desirable state. An idea of adaptation is derived from paper (Akzhalova and Poernomo, 2010) when the system adaptation happens by calling Reconfigure() selftransformation to make the system satisfy to desirable QoS characteristics which are pre-defined in QoSConstraints. In this work Reconfigure() generates Policy which is used then for a Binding appropriate Service by Service Request Controller. Therefore, Reconfigure() produces Policy as a product of the following transformation:

$$\textbf{Reconfigure} : System \times QoSConstraints \times SLA \rightarrow Policy \quad (1)$$

where reconfiguration of System is evaluated by its cost model defined by CostFunction which was described in previous section.

Every TimeStep when System violates QoSConstraints, Recongigure() defines Service.ID that has to be bound for each service request. We designate a candidate Service as $\{Policy(TimeStep) = ID, ID = 1,...,Numberof Services\}$.

To find best candidate service at each time step:

$BestPolicy(TimeStep) \in \{Policy(TimeStep) = ID, ID = 1,...,Numberof Services\}$

that satisfies to QoS constraints:

$$Constraints() \equiv true$$

and gives a minimum to an overall cost of the System:

$$System.CostFunction(Policy) \rightarrow \min, \quad (2)$$

where System changes its reconfiguration according to System.SystemConstraints():

$$SystemConstraints(TimeStep, Policy(TimeStep)) \quad (3)$$

4 EXPERIMENTS

The purpose of this experiment is to find out how proposed QoS and SLA aware infrastructure affects to performance of the cloud based system and relationship between cost and performance.

For simulation purposes, we take two Data Centers (DC) and four User Bases (UB). User Base is a group of users taken as single unit for the simulation purposes. User Bases are sources of traffic. An information about User Bases and Data Centers are given in Table 1, 2 and 3, respectively.

For the sake of simplicity, we have taken peak hours independent from regions. We implement analysis of handling multiple requests from different locations for duration period 30 days (approximately

Table 1: User Base specifications.

Name	Region	Requests per User	Fixed Data Size per Request	Avg Peak Users	Avg Off-peak users
UB1	2	60	100	1000	100
UB2	1	100	100	1000	100
UB3	1	50	100	1000	100
UB4	0	60	100	1000	100

Table 2: Data centers allocation.

Region	Cloud Id
North America	0
South America	1
Europe	2

Table 3: Data centers specifications.

Data Center	#VMs	Image Size (bytes)	Memory	Bandwidth
DC1	5	10000	1024	1000
DC2	5	10000	2048	1000

one month). We assume that there are 10 simultaneous users from single User Base requesting particular cloud service.

We consider case when number of requests from one user N varies between 50 up to 100 per second, a number of virtual machines m is 5 which are triggered by all user requests during an hour and k which is number of additional cloud providers is 2.

The proposed Policy is used across VMs in single Data Center. The "VM Cost" and "Data Transfer Cost" calculated by following formulas:

$VMCost = totalTime * CostperVM/Hr$

$DataTransferCost = totalData/(1024*1024) * DataTransferCost$

Input parameters for cost function is given in Table 4.

Applying MDE approach proposed in this work we calculate Data Center usage cost and best cloud service provider that can be given automatically to process user requests shown in Table 5 and Table 6.

Table 7 displays maximum, minimum and avarege achieved response time for different User Bases and their regions allocation shown in Figure 9. As it can be studied from the Table and examining all accommodated systems we have detected that the best variant in terms of minimum cost and response time was generated by policy that operates with services with highest number of physical machines.

We observe that total cost of the system decreases while handling services when the value of their capacities increases. Therefore, there is an inverse relationship between performance of existing resources (services) and cost and response time of the system. Figure 10 demonstrates loading of Data Centers that process requests according to automatic configurations done by proposed framework.

All measures were classified in interval based category which based on time intervals experienced by the entity or by events passing through the entity. An example of an interval based measure is the average waiting time of events at a certain entity.

Figure 9: Dynamics of response time.

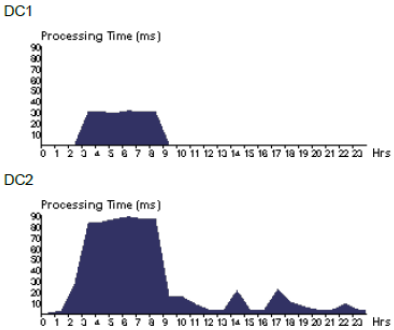

Figure 10: Dynamics of loading of the Data Centers.

Summarizing the case study, we have examined affection of changing number of requests, capacities

Table 4: Input parameters for cost function.

CostVM ($/hr)	CostMem ($/s)	StorCost ($/s)	TransferCost ($/Gb)	SwitchCost ($)	PenaltyCost ($/s)
0.001	0.0005	0.0001	0.1	3	0.1
0.003	0.0005	0.0002	0.6	3	0.1

Table 5: Data center usage cost.

Name	Cost per VM	Memory cost	Storage Cost	Data Transfer Cost	Physical Hardware Units
DC1	0.1	0.05	0.1	0.1	2
DC2	0.3	0.05	0.2	0.06	1

Table 6: Costs of using the clouds.

Data Center	VM Cost ($)	Data Transfer Cost ($)	Total ($)
DC2	21,792.24	66.062	21,858.302
DC1	7,263.61	109.848	7,373.458

Table 7: Response time estimated by regions and by User Bases.

Userbase	Avg(ms)	Min(ms)	Max(ms)
UB1	308.5	209.79	36,726
UB2	210.89	135.46	38,707
UB3	210.19	136.54	34,079.3
UB4	65.812	36.034	36,711.26

to the system characteristics such cost and response time:

- The experiments showed an automatic reconfiguration of the system that handles multiple requests from different User Bases according to the provided framework.
- The response time of the system differs significantly if the system process requests on more than one physical servers and therefore, with highest throughput.
- Employing OSPF gives better response time.

These investigations can produce essential recommendations when design cloud based system automatic resource allocation engine and struggle with workloads. The proposed Protocol and QoS and SLAbased reconfiguration can be used by cloud service providers to install dynamic SLA contract with service consumers.

5 DISCUSSION

(Cardellini et al., 2009) offers MOSES framework where SLA Monitor collects data about the average amount of requests from user, response time, cost of execution and reliability from both user and provider side. In (Cardellini et al., 2009) SLA Monitor notifies Adaptation layer about the fluctuation of current workload. However, the framework does not take into account the workload into the further model of adaptation. After receiving notifications and information about QoS attributes from Monitoring Layer, Adaption Layer is built over business process model and Adaptation Manager. The invocation of the business process causes generation of the new instance of the process which may differ depending on decision made by Adaption Manager. Therefore, Adaptation Manager solves new optimization problem for each new instance of the business process model. The optimization problem is represented by single objective optimization problem where the performance target is to make the distance between maximum values of QoS attributes and current ones as far as possible while meeting two types of constraints: negotiation between MOSES and each user and MOSES and providers. This problem was solved by linear programming approach. However, it has limitation in case of exponentially increasing number of concrete services. In this work we model the system where SLAs are measured and given from providers.

(Menascé et al., 2010) presents the framework that is a part of Model-Driven project SASSY (Self-

Architecting Software Systems). SASSY is a runtime of SOA self-architecting and re-architecting concept to meet functional and QoS requirements such as availability, execution time, and throughput.

In the SASSY framework, the domain expert has to specify desirable requirements using a visual activity-based language. Basing on these requirements The SASSY automatically generates a base architecture. This architecture will be optimized according to specified to QoS requirements through the selection of the most suitable service providers and application of QoS architectural patterns. Each service sequence scenarios (SSS) has own utility function that is related to one QoS metric and it is a subject to constraint. An overall utility function is used to adapt the whole architecture. The new architecture is created from the base architecture with the help of optimizing a utility function for the entire system. In our work we minimize overall cost function that reflects service providers's SLAs.

6 CONCLUSIONS

In this work we have offered distributed platform for QoS control and SLA based reconfiguration that allows to the cloud based system adapt at runtime depending on constantly changing QoS parameters by adjusting SLAs automatically by Service Request Controller engine. Therefore, service providers do not have to manually allocate service requests on fixed different servers. It will be done according to the predefined preferences and SLA contracts with other cloud service providers. The system reconfiguration will be done on QoS requirements in order to improve the performance of the system.

We have extended the meta-model proposed in previous works that selects best architecture by employing suitable OSPF optimization technique depending on requirement to the infrastructure. Therefore, it will provide QoS management of merged cloud systems.

The case study investigates how different parameters of the cloud system and SLA affect to the cost and performance. We have formulated recommendations for applying our approach to dynamically adapt cloud based system to desirable QoS characteristics.

REFERENCES

Akzhalova, A. and Poernomo, I. (2010). Model driven approach for dynamic service composition based on qos constraints. *Services, IEEE Congress on*, 0:590–597.

Bocchi, L., Fiadeiro, J., and Lopes, A. (2008). Service-oriented modelling of automotive systems. pages 1059 –1064.

Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., and Mirandola, R. (2009). Qos-driven runtime adaptation of service oriented architectures. In *ESEC/FSE '09: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 131–140, New York, NY, USA. ACM.

Cortellessa, V., Di Marco, A., and Inverardi, P. (2007). Non-functional modeling and validation in model-driven architecture. pages 25 –25.

Gönczy, L., Déri, Z., and Varró, D. (2009). Model transformations for performability analysis of service configurations. pages 153–166.

Groleat, T. and Pouyllau, H. (2011). Distributed inter-domain sla negotiation using reinforcement learning. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 33–40.

Keller, A. and Ludwig, H. (2003). The wsla framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1):57–81.

Krothapalli, N. K. C. and Deshmukh, A. V. (1999). Design of negotiation protocols for multi-agent manufacturing systems. *International journal of production research*, 37(7):1601–1624.

Le Sauze, N., Chiosi, A., Douville, R., Pouyllau, H., Lonsethagen, H., Fantini, P., Palasciano, C., Cimmino, A., Reichl, P., and Gojmerac, I. (2010). ETICS – QoS-enabled Interconnection for Future Internet Services. In *Proc. Future Network & Mobile Summit 2010, Florence*.

Menascé, D. A., Ewing, J. M., Gomaa, H., Malex, S., and Sousa, Jo a. P. (2010). A framework for utility-based service oriented design in sassy. In *WOSP/SIPEW '10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 27–36, New York, NY, USA. ACM.