Simulation-based Evaluation of an Intercloud Service Broker

Foued Jrad, Jie Tao and Achim Streit
Steinbuch Centre for Computing, SCC
Karlsruhe Institute of Technology, KIT
Karlsruhe, Germany
{foued.jrad, jie.tao, achim.streit}@kit.edu

Abstract—**The lack of common standards in a fast emerging Cloud computing market over the last years resulted in "vendor lock in" and interoperability issues across heterogeneous Cloud platforms. Therefore, the Cloud user is facing now a challenging problem of selecting the Cloud provider that fits his needs. A new promising research approach is the use of intermediate broker services to assist the user in finding the appropriate Cloud resources that satisfy his requirements. In this paper, we present a generic simulation framework based on the CloudSim toolkit for the validation and evaluation of a Cloud service broker deployed on an Intercloud environment. A unique feature of the framework is the integration of several state of the art technologies and standards, which makes it easy to deploy on real production Clouds. After presenting the framework fundamental architecture, we discuss in detail the solved implementation challenges. Finally, we present some initial evaluation results.**

Keywords—Cloud Brokering, Intercloud Computing, Simulation Environment, Cloud Interoperability, ClouSim Toolkit.

I. Introduction

The on-demand delivery of Cloud computing services over the Internet is now a needed reality rather than only a new marketing hype. Due to the fast emerging Cloud computing market over the last years, the number of Cloud service providers has significantly increased. On the other hand, "vendor lock in" issues and the lack of common Cloud standards hindered the interoperability across these providers. Thus, today the Cloud customer is facing a challenging problem of selecting the appropriate Cloud offers that fit his needs. Therefore, standardized interfaces and intermediate services are needed to prevent monopolies of single Cloud providers.

One of the promising use cases of the Intercloud vision defined in [1] is market transactions via brokers. In such a use case, a broker entity acts as a mediator between the Cloud consumer and multiple interoperable Cloud providers to support the former in selecting the provider that better meets his requirements. Another value-added broker service is the easy deployment and management of the user's service regardless of the selected provider through a uniform interface.

The lack of standardization across Cloud providers makes the deployment of Cloud service brokers on real production Clouds a challenging task for Cloud developers and researchers. Amongst others, many vendor compatible adapters are needed by the broker to interface the heterogeneous Cloud platforms. Furthermore, the evaluation of the broker using a real testbed is usually cost- and time-consuming, as a large number of Cloud resources is required to achieve realistic results. A more promising and cost-saving approach for the broker evaluation is the use of simulation environments.

Motivated by the above considerations, we present in this paper an extensible simulation-based framework to evaluate Cloud service brokers. The contribution from the developed framework is threefold: (1) It implements a Cloud service broker featuring automatic Service Level Agreement (SLA) negotiation and service deployment; (2) It enables through a standardized abstraction layer the monitoring and management of services deployed on heterogeneous Cloud providers while hiding their technical details; (3) It allows the easy integration and evaluation of custom resource matching policies.

The remainder of the paper is organized as follows: In the next section, we discuss prior works related to Cloud service brokering frameworks. We also identify how our work differs from related work. This is followed by the framework fundamental architecture in Section III. The simulation environment details are discussed in Section IV. In Section V and VI, we present and discuss initial evaluation results, respectively. Finally, we conclude the paper in Section VII with a brief summary and describe our future research directions.

II. Related Work

The idea of service brokering in Cloud is currently a subject for many research works.

A well-known research project is the Cloudbus toolkit [2] that defines a complete architecture for market-oriented Cloud computing. The three key components of this architecture are a Cloud Broker, a Market Maker and an InterCloud [3]. The Cloud Broker schedules applications on behalf of the user by specifying the desired Quality of Service (QoS) requirements, whereas the Market-Maker acts as a mediator bringing together Cloud providers and customers. It aggregates infrastructure demands from the Cloud Broker and matches them against the available resources published by the Cloud providers. The InterCloud provides a scalable federated computing environment composed of heterogeneous interconnected Clouds enabling the Intercloud resource sharing.

The above envisioned architectural framework is still under development. However, first experimental results with Aneka [4] and Amazon EC2 [5] based Clouds demonstrated that the market-oriented Cloudbus architecture brings benefits to user's application performance in optimizing the cost and execution time.

CloudAnalyst [6] is a graphical simulation tool built on top of the CloudSim toolkit [7], developed by the Cloud Computing and Distributed Systems (CLOUDS) laboratory at university of Melbourne to model and analyze the behavior of large social network applications. The Internet traffic routing between the user bases located in different geographic locations and the datacenters, is controlled in CloudAnalyst by a service broker that decides which datacenter should serve the requests from each user base based on different routing policies. The current version of CloudAnalyst implements three different routing policies, which are network-latency-based routing, response-time-based routing and dynamic-load-based routing. A CloudAnalyst simulation case study of the social network application Facebook [8] proved how load balancing managed by a service broker optimizes the performance and cost of large scale Cloud applications.

The EU funded OPTIMIS [9] project drives the development of a toolkit to optimize the full service lifecycle in the Cloud. Its proposed flexible multi-Cloud architecture includes a service broker that allows a decision making taking into account of many business aspects like trust, cost and risk. Although the toolkit is still not implemented, the conducted simulation experiments with real workload traces prove the benefits from the use of cost and risk aspects as elasticity policies in the decision making.

The work in [10] proposed an SLA-based Service Virtualization (SSV) architecture, which is built on three main components: a Meta-Negotiator responsible for agreement negotiations, a Meta-Broker for selecting the proper execution environment and an Automatic Service Deployer for service virtualization and on-demand deployment. The proposed service virtualization architecture has been validated in a simulation environment based on CloudSim using a real biochemical application as a case study. The simulation results showed the performance gains in terms of execution time from the SSV architecture compared to a less heterogeneous Grid meta-brokering solution.

Comparing the previous mentioned service brokering approaches, their implementation on real production Clouds is still ongoing and their current validations and evaluations are mostly based on simulation methodologies. The presented Cloud service broker framework in this paper is also implemented based on a simulation approach. However, its high-level generic architecture combined with the integration of state of the art Cloud technologies and standards prepares a realistic testbed for developers and researchers to easily test and evaluate service brokers before their deployment on real production Clouds. Moreover, the framework implements all the value-added broker services included in previous solutions like SLA negotiation, match making, service deployment and monitoring.

III. Framework Architecture

As shown in Figure 1, the framework architecture is composed of three main parts: the Client, the Cloud Service Broker and the Cloud provider Intercloud Gateway. The internal components of every architecture part and their provided functionalities are discussed in the following subsections.

A. *Client*

The Client provides Cloud users with an interactive user interface to submit their service requests to the broker by describing the functional and non-functional service requirements. Moreover, the user is able to manage and monitor the service after its deployment through a single management console. If the requested service requires the involvement of other services, a workflow engine could be deployed to assist users in building and executing complex Cloud services.

B. *Cloud Service Broker*

The Cloud Service Broker builds the heart part of our implemented framework by offering attractive value-added services to users. Its main task is to find the most suitable Cloud provider while satisfying the users' service requirements in terms of functional and non-functional Service Level Agreement parameters. Additionally, its high-level architecture design allows the deployment and monitoring of services on top of heterogeneous Cloud providers. More detailed descriptions on the internal broker design can be read in [11].

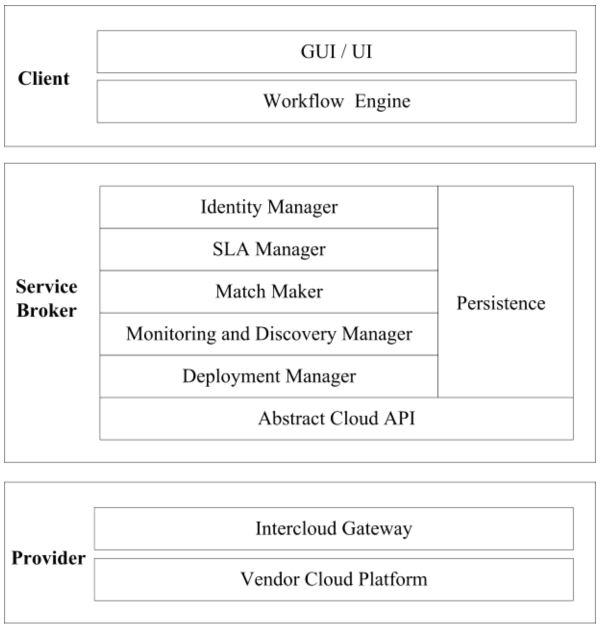

Figure 1. Framework architecture.

The different components of the broker and their roles are briefly described below:

- Identity Manager: It handles the user authentication and admission control.
- The SLA manager: It negotiates the SLA creation and handles the SLA provisioning.
- Monitoring and Discovery Manager: It queries resource information and monitors the SLA metrics.
- Match Maker: It selects the best Cloud providers for user requests using different matching algorithms.
- Deployment Manager: It deploys the service on the selected provider.
- Persistence: It stores broker specific data (e.g., monitoring, SLA templates and resources data).
- Abstract Cloud API: A standard abstract API used to manage Cloud resources on different Cloud providers.

C. *Provider Intercloud Gateways*

The Intercloud Gateway is the key component of our framework hosted on the provider side to interface the vendor Cloud platform. It acts as a standardized service frontend for the Cloud provider and adds the needed abstraction layer to interact with the broker. Its main role is to provide the broker with common management and monitoring interfaces while hiding the internal provider policies.

IV. Simulation-based Implementation

We implemented a simulation environment for the framework presented in the previous section. This allows us to validate and evaluate a Cloud service broker without the setup of a testbed with real Cloud providers, which is extremely time- and cost-consuming. The implementation details and the simulation flow are described in the next subsections.

A. *Simulation Environment*

The simulation environment for the Cloud service broker framework built on top of the CloudSim 2.2.1 simulation toolkit is depicted in Figure 2. In the following subsections we go through all the implemented components by describing the used technologies and tools.

1) *CloudSim Toolkit*

CloudSim is a scalable open source simulation tool offering features like support for modeling and simulation of large scale Cloud computing infrastructures including datacenters, brokers, hosts and virtual machines (VMs) on a single host. In addition, the support for custom developed scheduling and allocation policies in the simulation made CloudSim an attractive tool for Cloud researchers. Additional information about CloudSim can be found in [12].

In our simulation environment CloudSim is used to model large scale and heterogeneous Cloud providers. This allows us, for the purpose of evaluation, to easily configure the amount of Cloud provider resources accessible by the broker. However, some CloudSim extensions are needed to allow the dynamic creation, destroying and monitoring of the VMs during simulation runtime and therefore to enable the automatic service deployment in the broker.

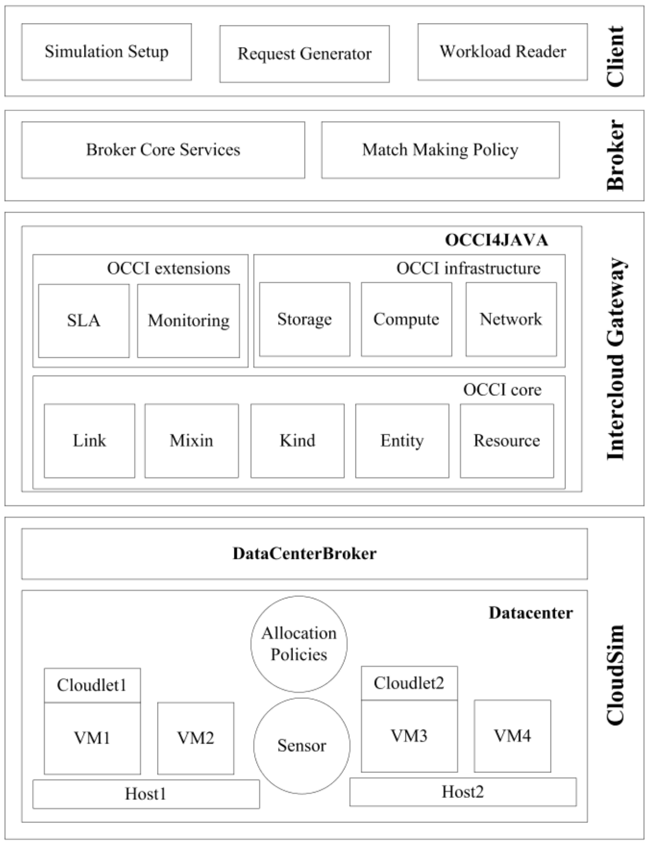

Figure 2. Simulation environment.

2) *Cloud Service Broker Implementation*

Until the writing of this paper, most core broker services including the Deployment Manager, the Match Maker and the Monitoring Manager have been fully implemented. The SLA Manager is currently under development. Furthermore, two persistence classes named ServiceRegistry and ProviderRegistry are used to store and query all the service and provider data during the simulation.

While looking for an abstract Cloud API to access different Cloud platforms, we found that the Open Cloud Computing Interface specification (OCCI) [13] is the most suitable for our framework. OCCI is an extensible specification for remote management of Cloud infrastructures, allowing the development of interoperable tasks over heterogeneous Clouds. The current OCCI specification, focusing on IaaS Cloud provisioning, defines three abstract resource types, which are compute, storage and network. All the operations on resources can be requested on a REST manner over HTTP methods (GET, POST, PUT and DELETE). The use of OCCI as abstract Cloud API allows the broker to act as OCCI client against the Intercloud Gateway, which runs as OCCI-server on the provider side.

The implemented Match Maker functionality of the broker is extensible enough to permit the easy integration of custom resource matching policies. In order to demonstrate this feature, we implemented the following primitive match making policies:

- RandomCloudMatcher: It selects randomly a provider regardless of the users' service requirements.
- FunctinalSLACloudMatcher: It selects the provider that fits all the functional SLA service requirements.
- LocationAwareCloudMatcher: It selects a provider located at the same region given in the service request.
- CostAwareCloudMatcher: It selects the cheapest provider below a given cost limit.
- HybridCloudMatcher: It combines both functional SLA and location-aware matching.

3) *OCCI-based Intercloud Gateway*

In order to simulate the Intercloud Gateway component serving as standard service frontend for Cloud providers, we implemented, based on the open source Java implementation for OCCI called OCCI4JAVA [14], an OCCI frontend for CloudSim. In this way, the entire communication between broker and providers is forwarded to the native CloudSim DatacenterBroker class through standard OCCI-interfaces. In contrast to the OCCI specification, as CloudSim simulations usually run on one host, the broker communicates with the Intercloud Gateway through simple Java object calls instead of using the defined REST-like methods. Furthermore, we extended OCCI4JAVA with an OCCI monitoring mixin to allow the broker to query resource properties like datacenter static information (e.g., location, supported OS, CPU architecture) and current monitoring metrics values from CloudSim.

4) *Request Generator*

The simulation-based evaluation of the broker requires the submission of real world service requests by the client to achieve valuable evaluation results. Thus, we implemented a service Request Generator helper class that continuously generates VM provisioning requests similar to real Amazon EC2 compute instances at a configurable rate. Some sample VM requests are provided in the next section.

B. *Simulation Flow*

The needed simulation flow to process the incoming client service requests to the service broker is illustrated by the flow diagram in Figure 3.

The simulation is done as follows: In a first step, CloudSim is initialized according to the desired simulation scenario. Then, the Request Generator starts to generate continuously VM provisioning service requests with a variable request arrival rate. All the request and provider data are maintained in the corresponding ServiceRegistry and ProviderRegistry classes during the simulation. The broker, after receiving the request, asks the Match Maker, if the service can be deployed with the specified requirements. For this, the Match Maker starts a match making process to find the best suitable provider by matching the gathered resource information from the Monitoring Manager with the service requirements and by applying the pre-configured matching algorithms. Upon the existence of a match, the service is automatically deployed and the requested VM is created and started on the selected CloudSim datacenter with the modeled workload traffic (Cloudlet). During the execution time, the VM status is queried periodically by the Monitoring Manager until the VM is destroyed. If none of the providers can be matched, the request is discarded by the broker.

All the aforementioned simulation steps are repeated until reaching the preset maximum number of requests or simulation time limit. In this case, the simulation is terminated and the output results are displayed in the Client.

V. EVALUATION RESULTS

In this section we discuss first evaluation results acquired using the previous implemented simulation framework. We describe in the following subsections the experimental setup and then present the evaluation results.

A. *Experiemental Setup*

In order to model heterogeneous Cloud providers, we configured six heterogeneous CloudSim datacenters. Each datacenter has a unique identifier (ID) and is located into a different geographical zone. As shown in Table I, we define six different compute zones presenting the six world continents. Each zone has been given a unique code. The detailed configuration for each datacenter is gathered in Table II. The six datacenters have different pricing policies and can support one of two defined operating systems (Linux or Windows) and CPU architectures (x86 or x64). Furthermore, each datacenter is made up of 50 hosts, which are equally divided between two different host types. As can be seen in Table III, the used hosts' setup allows at least the

TABLE I. COMPUTE ZONES

Zone	***North America***	***South America***	***Europe***	***Asia***	***Africa***	***Austr-alia***
Code	0	1	2	3	4	5

TABLE II. DATACENTERS CONFIGURATION

Name	**Datacenter Configuration**				
	ID	***OS***	***Arch***	***Region Code***	***Cost $/hour***
Provider_A	0	Linux	x64	0	0.3
Provider_B	1000	Linux	x64	0	0.45
Provider_C	2000	Windows	x64	2	0.75
Provider_D	3000	Linux	x64	2	0.55
Provider_E	4000	Linux	x64	3	0.15
Provider_F	5000	Windows	x86	5	0.04

TABLE III. HOSTS SETUP

Host Type	**Host Configuration**				
	CPU MHZ	***Cores number***	***RAM GB***	***Bandwidth Gbit/s***	***Storage TB***
Xeon 3040	1860	2	4	1	1
Xeon 3075	2260	2	8	1	1

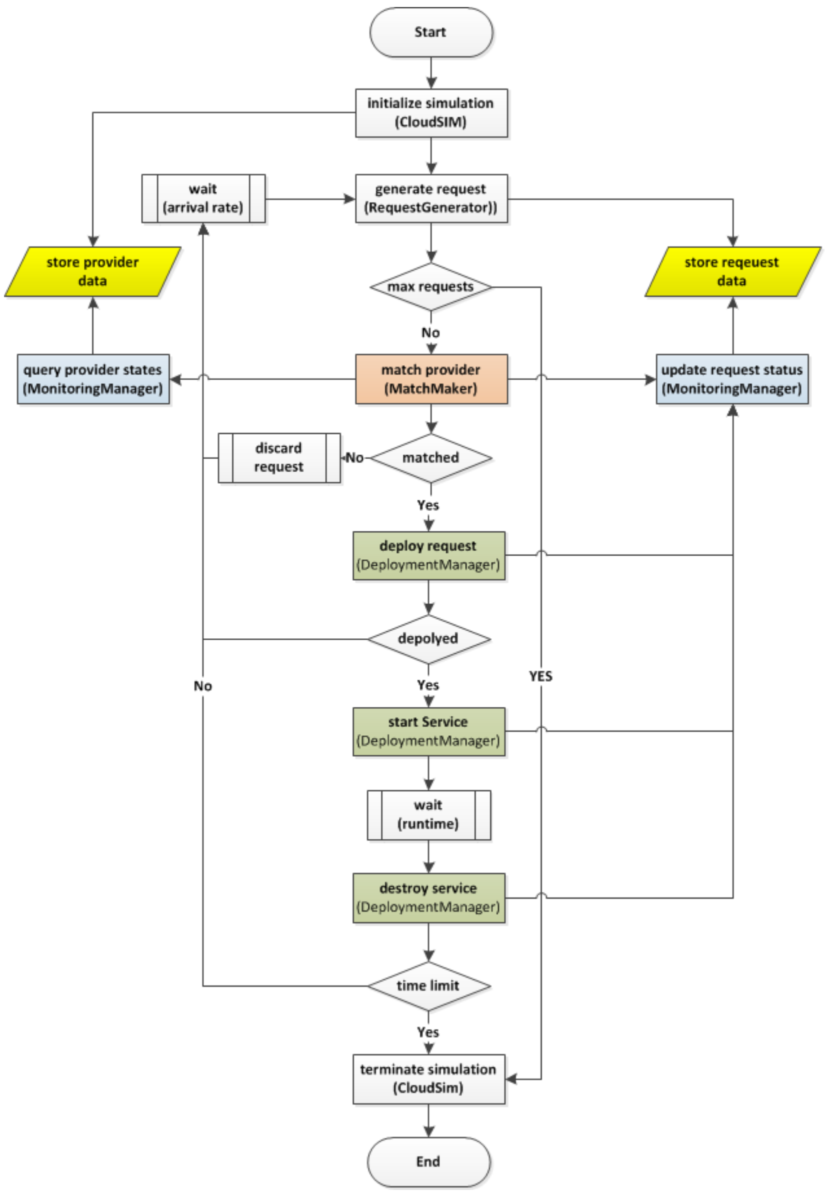
Start
initialize simulation (CloudSIM)
wait (arrival rate)
generate request (RequestGenerator))
store provider data
store reqeuest data
max requests
No
query provider states (MonitoringManager)
match provider (MatchMaker)
update request status (MonitoringManager)
discard request
No
matched
Yes
deploy request (DeploymentManager)
depolyed
No
Yes
YES
start Service (DeploymentManager)
wait (runtime)
destroy service (DeploymentManager)
time limit
Yes
terminate simulation (CloudSim)
End

Figure 3. Simulation flow.

deployment of one VM instance per host.

All the experiments are done on a notebook with CPU Intel Core i5 560M 2.67 GHZ, RAM 4 GB and using Windows 7 operating system. The default CloudSim simple VM provisioning policy is used as internal datacenter scheduling policy. This policy allocates VMs to the host with most free cores. In order to permit the dynamic sharing of CPU cores among VMs, we configured CloudSim to use a time-shared VM scheduler policy.

B. *Initial Results*

We conducted a first experiment to evaluate the broker scalability. We continuously generate VM service requests (at a random rate varying from 0 to 60 seconds) and let the broker randomly select a provider from the six datacenters and then deploy the VM on it. The generated VM requests are equally distributed between four Amazon EC2 instance types and require Linux as operating system and x64 CPUs. Table IV gives the specific requirements of each VM type.

TABLE IV. VM REQUEST TYPES

VM Type	**Host Configuration**				
	CPU GHZ	***Cores number***	***RAM GB***	***Region Code***	***Cost $/hour***
CPU high	2.5	2	1.7	0	0.17
large	2	2	7.5	2	0.34
small	1	1	1.7	3	0.085
micro	0.5	1	0.63	4	0.02

TABLE V. BROKER DEPLOYMENT PERFORMANCE (%)

Provider number	Requests number					
	50	*250*	*500*	*1000*	*1500*	*2000*
1	98	78	44.4	28	22.1	17.8
3	100	81.6	78.6	61,8	44	36.3
6	100	98.3	82.8	78.6	77.2	62.1

While maintaining the number of datacenters constant, we measured the deployment rate, which is defined as the percentage of successfully deployed VMs, by varying the request number from 50 to 2000. We repeated the same experiment by decreasing the number of datacenters from six to three and then to only one. The results presented in Table V after one day simulation time, show that the broker deployment rate scales well with the increasing number of service requests and Cloud providers.

We conducted another experiment to evaluate the match performance of the four implemented primitive matching policies. We repeated the previous experiment using all six datacenters and by changing each time the matching policy and then we measured the match rate, defined as the percentage of successfully matched requests.

As depicted in Figure 4, when using cost or location as matching policy the match rate remains constant at 75 %, as the requested cost limit and location for the micro VM instance type usually has no match. However, with the functional SLA and hybrid matching policies the match rate decreases continuously with the rising service demand due to the limited capacity of the provided datacenter resources.

VI. DISCUSSION

The previous experiments show that an increase of the number of concurrent providers results in more resource heterogeneity and therefore improves the broker match rate. Furthermore, the results prove that the accuracy of the queried monitoring information by the Monitoring Manager heavily impacts the performance of the matching policy, especially for the functional SLA matching.

In fact, the support of more than one SLA parameter in the matching increases the customer satisfaction, but at the cost of a low match rate. Thus, the matching algorithm should optimize this trade-off by modeling the dependency between the customer utility function and his requested functional and non-functional SLA parameters, while considering the current provider monitoring information.

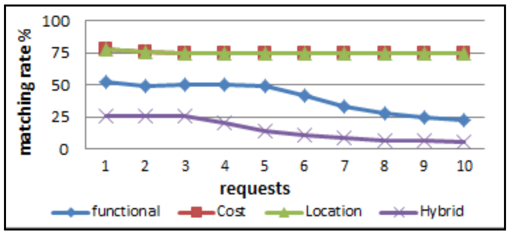

Figure 4. Broker matching performance.

VII. CONCLUSIONS AND FUTURE WORKS

The deployment and evaluation of intermediate broker services on production Clouds is today a challenging task due to the lack of interoperability and the heterogeneity in current Cloud platforms.

In this paper, we described the fundamental architecture and the implementation details of a simulation-based framework used to evaluate a Cloud service broker. We presented also the first simulation results in evaluating the broker scalability and match making policies.

In our future work, we will use the simulation framework to investigate and evaluate more complex SLA-aware match making algorithms to improve the broker matching performance. Furthermore, we will investigate the use of real workload traces instead of using generated requests to get more realistic results.

REFERENCES

[1] Golobal Intercloud Technology Forum GICTF, “Use Cases and Functional Requirements for Inter-Cloud Computing,” White paper, August 2010.

[2] R. Buyya, S. Pandey, and C. Vecchiola, “Market-Oriented Cloud Computing and the Cloudbus Toolkit,” in Large Scale Network-centric Computing Systems, March 2012, in press.

[3] R. Buyya, R. Ranjan, and R. N. Calheiros, “InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services,” in ICA3PP 2010, 10th International Conference on Algorithms and Architectures for Parallel Processing, pp. 13–31, 2010.

[4] “Aneka enterprise Cloud platform,” [online], March 2012, http://www.manjrasoft.com.

[5] “Amazon Elastic Compute cloud EC2,” [online], March 2012, http://aws.amazon.com/ec2.

[6] B. Wickremasinghe, R. N. Calheiros and R. Buyya, “CloudAnalyst: A CloudSim-based Visual Modeller for Analysing Cloud Computing Environments and Applications,” in AINA 2010, 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 446–452, April 2010.

[7] “CloudSim Toolkit 2.1.1,” CLOUDS Lab, Unversity of Melbourne, [online], http://www.cloudbus.org/cloudsim/.

[8] Facebook, [online], March 2012, http://www.facebook.com.

[9] A. J. Ferrer, et al., “OPTIMIS: A Holistic Approach to Cloud Service Provisioning,” in Future Generation Computer Systems, vol. 28, pp. 66–77, January 2012.

[10] A. Kertesz, G. Kecskemeti and I. Brandic, “Autonomic SLA-aware Service Virtualization for Distributed Systems,” in PDP2011, 19th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 503–510, February 2011.

[11] F. Jrad, J. Tao and A. Streit, “SLA Based Service Brokering in Intercloud Environments,” in CLOSER 2012, 2nd International Conference on Cloud Computing and Services Science., pp. 76–81, April 2012.

[12] R.N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose and R. Buyya, “CloudSim: a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms,” in Journal of Sotware: Practice in Experience, vol. 41, pp. 23–50, January 2011.

[13] “Open Cloud Computing Interface specification OCCI,” OCCI-WG, [online], March 2012, http://www.occi-wg.org.

[14] “OCCI4JAVA JAVA-based OCCI Implementation,” [online], March 2012, https://github.com/occi4java.