J2CBROKER: A Service Broker Simulation Tool For Cooperative Clouds

Maurizio Giacobbe
Department of Engineering
University of Messina
Contrada Di Dio - 98166,
Messina
mgiacobbe@unime.it

Riccardo Di Pietro
Department of Engineering
University of Messina
Contrada Di Dio - 98166,
Messina
rdipietro@unime.it

Carlo Puliafito
DIEEI
University of Catania
Viale Andrea Doria 6 - 95100,
Catania
carlopulia@gmail.com

Marco Scarpa
Department of Engineering
University of Messina
Contrada Di Dio - 98166,
Messina
mscarpa@unime.it

ABSTRACT

The Internet and digital technologies are transforming our world, but existing barriers, mainly due to obsolete Information Technologies (IT), lead citizens to miss out on goods and services, enterprises and start-ups to limit their horizons, and businesses and governments to cannot fully benefit from digital tools. Recently, *Cloud computing* emerged as hot topic in IT, both in industrial and academic area, in order to overcome the above barriers. Its use in large scale distributed infrastructure, platform or software services is motivated by the possibility to promote a new economy of scale in different contexts. Such scenario demands timely, repeatable, and controllable methodologies for evaluation of algorithms, applications and policies before the development of Cloud services or products, especially to achieve a good compromise between several performance indicators. To this end, **simulations-based environments** allow to evaluate the hypothesis prior to the software development, thus reducing the risk of economic losses, scarce *Quality of Service* or *Quality of Experience*. In this paper we present and discuss a simulation-based approach for **Cloud Brokerage ecosystems**. More specifically, we propose the **J2CBROKER** *Simulation Tool*, mainly based on *JAVA* and *JavaScript Object Notation (JSON)* technologies. Its architecture, functionalities and technological choices are discussed and motivated. Moreover, we present a *case of study* to evaluate the goodness of the proposed approach.

Keywords

Broker, Cloud Brokerage, Cloud Computing, Cooperative

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266614

Clouds, Digital Single Market, Key Performance Indicator, JAVA, JavaScript Object Notation, Simulation Tool.

1. INTRODUCTION

Nowadays, Cloud computing represents the reference technology for delivering distributed, scalable, heterogeneous, reliable, fault-tolerant, and sustainable computational services. They are generally presented as Infrastructure, Platform, or Software as a Services (IaaS, PaaS, SaaS). However, recently the XaaS acronym emerged as a collective term signifying "X as a Service", "anything as a Service" or "everything as a Service", thus referring to services that are delivered over the Internet rather than locally or on-site, especially in presence of the emerging *Internet of Things (IoT)* paradigm. Moreover, services may be offered by *Cloud Service Providers (CSPs)* in private Data Centers (DCs), i.e. by *private Clouds*, or they can be commercially offered for customers, i.e. by *public Clouds*, or yet it is possible that both public and private Clouds are combined forming *hybrid Clouds*.

The above wide set of Cloud architectures demands timely, repeatable, and controllable methodologies for evaluation of algorithms, applications, and policies before the development of Cloud services or products, especially with the increasing demand for energy-efficient *IT* solutions. Since the use of real testbeds limits the experiments to the scale of the testbed, thus making the reiteration of results an extremely difficult undertaking, alternative approaches need to be considered both for Clouds and traditional IT systems. Among these possible alternatives, **simulations tools** allow researchers and practitioners to evaluate working hypothesis before the software development. In this manner it is possible to set environment variables and parameters defining models, reproducing tests and providing results (textual and/or graphical) to analyze. Specifically, the use of a simulation-based approach is fundamental for Cloud environments, because access the infrastructure incurs payments in real currency (*pay-as-you-go* system) At the customer side, simulation tools can offer significant benefits, for example by allowing Cloud customers to test their services

in repeatable and controllable environment, without paying Cloud services. At the provider side, instead, simulators can allow evaluation of different scenarios where to allocate computational resources under varying performances, workload conditions, and monetary cost distributions. CSPs can proceed in their *business analysis* thus to optimize the resource access cost with a special focus on improving profits. In the absence of such simulation-based environments, both Cloud customers and providers risk to commit serious errors of assessment, or to refer to non-objective evaluations resulting in inefficient service performance and economic losses.

2. MOTIVATIONS

Within the *Digital Single Market* [1] *European Commission* priority, Cloud computing plays a key role through the *data-driven innovation* initiatives, *data ownership, access* and *usability ownership, portability of data* and *switching* of service providers. In this complex context, customers' discovery of the services and selection of the one which best suits their needs is not a trivial issue and might be very time-consuming and/or ineffective. Moreover, customers need to be assisted during this process. A **Cloud Service Broker (CSB)** might be the solution to this problem. Essentially, a CSB is an additional computing layer which acts as an intermediary between service customers on one side and service providers on the other. *Gartner* [14], the world's leading information technology research and advisory company, identifies three areas in which Cloud brokerage might play an important role towards service customers, but also service providers. The first area is *aggregation*, that is to manage multiple services, possibly from different providers, presenting them as an unified service. Complex relationships and agreements among providers define the level of criticality. The second area is *integration* to make applications, which are independent at first, work nicely together and cooperate in order to fulfill the customer's needs. The third and last area is the *customization* one, which consists in the tweaking of services in order to best suit users' needs.

Other applications of CSB, which still come under the concept of service selection, are the *ranking* of services according to parameters provided by users (e.g. services ordered by cost) and the *selection* of the best DC (or site), among the N available, to execute a certain job.

Cloud Brokerage Enablers are software platforms which enable Cloud Brokerage. According to *MarketsandMarkets* [2], a market research firm, the Cloud Service Brokerage and Enablement market size is estimated to grow from USD 7.44 billion in 2016 to USD 26.71 billion by 2021. Some of the most important CSB companies are, in alphabetical order: *Appirio*, *ComputeNext* and *Dell Boomi*.

3. RELATED WORKS

Due to the great interest and importance played by CSB, many researches have been carried out in this field, surveying the possible approaches and algorithms for the service selection [16], [17], [13], but also the Cloud simulators which can be used to evaluate a CSB's performances [15]. In fact, the evaluation of performances in a real Cloud environment is too cost-and-time-consuming. This is why, very often, researchers and businesses prefer to use simulation tools to evaluate their proposals.

Probably, **CloudSim** [7] is the most popular and complete framework for the modeling and simulation of Cloud environments. It was developed at the *CLOUd computing and Distributed Systems (CLOUDS) Laboratory*, in the *University of Melbourne, Australia*. It is open source, entirely written in *JAVA* and provides basic classes for modeling DCs, users, brokers, computational resources, policies and virtual machines. CloudSim is built on top of another open source framework, named **GridSim** [6], used for large scale Grid systems and P2P networks.

Thanks to its success, CloudSim has been extended by researchers and thus other products using it as their core have been developed. The most important example of these is **CloudAnalyst** [18], which is a JAVA based simulation tool. It was developed for a precise purpose, which is that of dealing not just with a tool, but with an easy to use tool.

GreenCloud [11] is an open source simulation environment built as an extension of the *NS2* network simulator, which is written in *C++* and *OTcl*. The peculiarity which distinguishes this environment from all the others is the focus on the energy consumed by all the computing and communication components of a DC in a simulated Cloud environment. GreenCloud is a packet level simulator, meaning that whenever a packet has to be transmitted, all the relative protocol processing is performed. On the other hand, CloudSim and CloudAnalyst are event-based simulators: they avoid to process packets individually and they capture the overall effect of interaction instead. The result is that even if GreenCloud is more accurate, it is slower in simulating and is able to simulate systems with a lower number of nodes. Unlike CloudAnalyst, GreenCloud does not have a GUI for an easy simulation setup, nor for the representation of the simulation results as tables or charts. It uses the Nam animation tool to visualize the simulated topology and the packet flow after the simulation is completed.

iCanCloud [12] is an open source simulation platform entirely written in C++ and developed as an extension of the *OMNET* network simulator. The main purpose of iCanCloud is to estimate the trade-off between costs and performances, thus to help users in to optimize it. iCanCloud has a feature that the previously discussed environments do not have: if there is a cluster of nodes available to carry out an experiment, it is possible to perform a parallel simulation among them.

However, even if many simulator tools that can be used for studying Cloud systems, we can not establish what is the best simulator to use because the evaluation depends upon actual requirement.

4. THE J2CBROKER SIMULATION TOOL

Based on the above considerations, we thought of a tool capable of simulating different cooperative Cloud Brokerage scenarios across different metrics and evaluation criteria included in *models* such as *sustainability* and *availability* in a single "multi-criteria" model.

4.1 Architecture and Technological Choices

We thought of a tool capable of managing **JSON** documents and written in **JAVA**: hence the name *Java Json Cloud BROKER (J2CBROKER)*. The main goal is to provide a simulation-based tool that: *i)* dynamically manages JSON documents as inputs simulating both requests and offers by CSPs; *ii)* calculates the best choices (i.e., offers) on the basis of specific parameters through different multi-

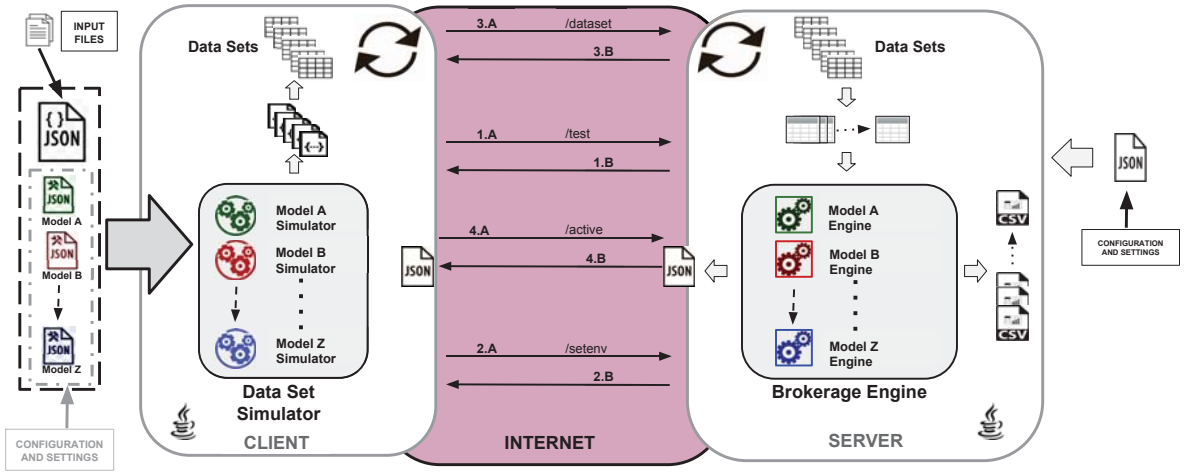

Figure 1: A general architecture of the *J2CBROKER* Simulation Tool.

criteria engines (i.e., multi-criteria algorithms implemented in JAVA language); *iii)* provides the resulting best offers of its calculation as outputs, both in the form of JSON documents and *on-screen* (results can be also provided in many other forms, such as *CSV*).

JSON is a lightweight, text-based, language-independent data interchange format derived from the *ECMAScript Programming Language Standard*. It defines a small set of formatting rules for the portable representation of structured data [3]. It has a simple syntax, which results, for example, in less "markup" overhead, more compact and more readable if it is compared to the *eXtensible Markup Language (XML)* alternative [9].

The J2CBROKER **general architecture** is shown in Figure 1, where we introduced the concept of **model**. A model is essentially a JSON file which contains all the *input* metrics describing the basic characteristics of a given Scenario.

The JAVA client-server architecture uses a stateless *RESTful* approach for its communication. The client and the server can operate on different machines across the Internet. The server is modeled as a perpetual process that waits for clients to receive requests, and then processes them. The server offers resources through *REST API*. The use of the JAVA *object-oriented, class-based* and *general-purpose* computer programming language [10] is an advantage. It allows to run the same program on many different systems by being platform-independent at both the source and binary levels. Moreover, it is ideal to create modular programs and reusable code. Before starting their activities, both the client and the server have a mandatory JSON configuration file called *json-conf-file* that the user have to fill in with the appropriate information. At the client side, the configuration file contains besides the client/server configuration settings, the specific information of the Model that the user wants to use during the simulation. At the server side, this file contains only the server configuration settings. These two configuration files allow the user to set up the client-server architecture and to dynamically configure the behavior of the simulation tool at each new simulation session.

For each model, the architecture provides two specific components: the related Model Simulator at the client side, and the related Model Engine at the server side. More specifically, referring to Figure 1, we mainly distinguish two blocks: the **Data Set Simulator** at the client side and the **Brokerage Engine** at the server side, both containing several Models (respectively Simulators and Engines).

4.1.1 The Data Set Simulator

The Data Set Simulator is the "core" of the client application. It consists of a modular structure which contains different Model Simulators. The result is a general purpose tool which allows anyone to create and connect his own Model Simulator. Each Model Simulator implements different specific behaviors. Those latter depend on both the directives received from the *json-conf-file* and the characteristic dedicated for the simulation. If the user uses the client-server application according to the directives of the Random Simulation Mode, the Data Set Simulator creates specific Data Sets according to a dedicated Model. Otherwise (i.e., according to the directives of the Guided Simulation Mode) the Data Set Simulator gathers and parses the information from the JSON absolute paths listed inside the *json-input-list-file*. In any case, the expected behavior at the server side will be the same. It will store all the Data Sets and, when it will receive the *active request*, it will elaborate the Data Sets according to the Model Engine predetermined in the simulation. Finally the server will send a JSON file to the client with the result of the calculation at the Brokerage Engine. In such a context, each Data Set represents a simulated offer by a CSP at a specific Cloud site.

4.1.2 The Brokerage Engine

The "Brokerage Engine" represents the "core" of the Server application. It consists of a modular structure that contains different Model Engines. This is important because it makes

this client-server application a general purpose tool, which allows anyone to create and connect his own Model Engine. However, each Model Engine implements different specific behaviors. Those latter depend on both the directives received from the client through the HTTP POST "/setenv" request, and on the characteristic of the dedicated brokerage scenario.

4.1.3 The Client/Server Communication

The communication phase between the client and the server is done in four different steps (see Figure 1). First of all, the client verifies if the server is alive by sending to it an *HTTP POST* request for the resource "/test" **(1.A)**. If the server returns a response containing the status code equal to 200 **(1.B)**, which means that the test is successfully done, then the client can move to the next step **(2.A)**.

In order to set some server side environment parameters, the client sends an *HTTP POST* request to the server for the resource "/setenv" and specifies those parameters inside the *Headers* of the HTTP request **(2.A)**. If the server returns a response that contains the status code equal to 200 **(2.B)**, which means that the set-environment request is successfully done, then the client can move to the next step **(3.A)**.

In order to transfer all the Data Sets to the server, the client uses one *HTTP POST* request for each Data Set. All these requests are for the resource "/dataset" **(3.A)** and each one will contain all the information about a particular Data Set the client wants to transfer at that moment. All these information are parameters stored as Headers inside the HTTP request. If the server returns a response that contains the status code equal to 200 **(3.B)**, which means that the dataset request was successfully done, then the client will send another dataset request, and so on, until the end. Then the client can move to the next step **(4.A)**.

To start the server side processing phase, the client sends *HTTP POST* request to the server for the resource "/active" **(4.A)**. When the server will complete the processing with success, it will return a response with the status code equal 200 and the JSON file containing the output of the processing phase **(4.B)**.

4.1.4 Simulation Modes

The J2CBROKER can work in two different modes: the **Random Simulation Mode** and the **Guided Simulation Mode**. Both the above simulation modes are part of the setting at client-side.

During the *Random Simulation Mode*, the client reads the information of the specific Model described inside the related *json-conf-file*. According to the reported information, the client simulates the Data Sets and takes care of sending them, one per time, to the server, along with other directives addressed to the related Model Engine. Once the server received all the information needed, it will elaborate them and returns the output through a new JSON file.

In order to be run in the *Guided Simulation Mode*, the client needs the above-mentioned *json-conf-file*, along with another mandatory JSON configuration file called *json-input-list-file*. This latter needs to be filled with the absolute paths of all the JSON files representing the Data Sets that the user wants to use in this guided Simulation. The client reads all the absolute paths indicated inside the *json-input-list-file*, understands what Model they are related to and makes a parsing operation to extract all the information needed to create the correspondent Data Sets. Therefore, the client sends one Data Set per time, along with other directives addressed to the related Model Engine, to the server. At this time, the server will do the same processing as in the Random Simulation Mode.

5. CASE OF STUDY: SUSTAINABILITY-ORIENTED MODEL

In this Section, we present a *case-of-study* to prove the goodness of the proposed methodology. By referring to Figure 1, we introduce a **sustainability-oriented Model** to make the best choice in *resource allocation* thus to push down environmental pollution (sustainability), and contemporary taking into account the availability and the **monetary cost** criterion in a *multi-criteria approach*.

We started the J2CBROKER application server on a *Virtual Machine* equipped with *Ubuntu Server 14.04* and hosted in a *IBM BladeCenter LS21* at the *Cloud Laboratory DC - University of Messina*. We tested the application server by running several application clients, hosted in different machines both inside and outside the above-mentioned DC.

5.1 The Sustainability Metric

We refer to *sustainability* as part of the concept of *sustainable development* [8] that is "the development that meets the needs of the present, without compromising the ability of the future generations to meet their own needs" [4]. For our energy-aware purposes, in the following we express sustainability through several sub-metrics (Table 1 to follow) which are generally used to define "how green is a DC".

5.2 The Monetary Cost Criterion

Monetary Cost is a quantifiable criterion that addresses customers and organizations in their business. By referring to IT services it is generally expressed in $\$/h$ (i.e., dollars-per-hour) or $\$/GB$ (i.e., dollars-per-GigaByte). Usually providers offer instance placement services with a fixed price in the service maintenance time at a site. Therefore, we can express cost for an instance i at a site node s, as follows:

$$cost_{s,i} = service\_price_{s,i} * \Delta_t \quad (1)$$

where Δ_t is the *running time* of the $i-th$ instance at the $s-th$ site node.

5.3 Scenario

The proposed tool simulates a scenario where the main goal is to reduce *carbon dioxide emissions* (i.e. the *CO2*) through a Cloud brokerage ecosystem, where *Cloud Service Providers (CSPs)* cooperate in a *centralized brokerage environment* to run *instance workloads* at the most convenient Cloud sites. An instance is a temporary virtual server that needs to be allocated in order to run services. More specifically, we developed a sustainability-oriented Model Data Set at client side and a sustainability-oriented Model Engine at server side.

5.3.1 Roles

The entire simulated process mainly consists into three roles (*applicant (A), brokering (B)* and *offering (O)*).

Role A. When a Cloud provider receives some instances to run it generates a request to B for computation, and

Sustainability-oriented Model Data Set	
Service Data Set	
Parameter	**Range**
Instance workload (watts)	200-300
Power basic (watts)	100
Running time (hours)	10, 24, 360, 750
Number of instances in each request	1, 10, 20, 50
Number of instances in each offer	12, 14, 16, 18, 20
Availability (%)	99.90-99.99
Service price ($/h)	0.007-0.112
Sustainability Data Set	
Parameter	**Range**
ITEU	0.3-0.6
ITEE	0.1-3.9
PUE	1.4-2.3
GEC	0.0-0.003
CDIE (kgCO2/kWh)	*source: https://www.ipcc.ch*

Table 1: The Sustainability-oriented Model Simulator Data Set.

cooperates within the ecosystem to determine where the load should run.

Role B. At each new event concerning the status of requests or offers, it gathers the requests from A profiles and evaluates the offers (sent by the O profiles) by using its Model Engine (i.e. the algorithm). Finally, the resulting best offers, where an instance has to run, are presented to the A profiles.

Role O. Each CSP creates one or more offers, each one based on sustainability, availability and cost factors at the involved site. Therefore, it sends its offer to B.

The J2CBROKER allows the user to simulate the abovementioned roles' tasks by configuring and executing the sustainability oriented Model Data Set and the sustainability oriented Model Engine, where a developed energy-aware algorithm runs.

5.3.2 The Sustainability-oriented Model Simulator

The proposed simulation environment presents a modeling of both service and Cloud site, i.e. the **Model Data Set**, thus to provide *input* data for the related Model Engine. Each offer is modeled by a *JSON* document, i.e. the Model Data Set, which includes two main collections: the first one defines a service data set describing the service parameters and among these availability and cost; the second one defines the sustainability metrics and factors at each Cloud site.

The service data set is obtained from a survey on several "top" providers of IT technologies (e.g., Dell blade servers), Cloud services and solutions (e.g., Amazon Web Services (AWS)). The sustainability data set, instead, is based on the measurement results of a real scenario, from the METI Japan project [5] on enhancing the energy efficiency and the use of green energy in DCs.

Table 1 shows the Model Data Set: the simulator selects a random value between the range set for each metric in order to characterize each offer by its sustainability, availability and monetary cost values.

5.3.3 The Sustainability-oriented Model Engine

The Brokerage Engine consists of a modular structure that contains different Model Engines. Among these, in the presented case of study, we refer to the sustainability-oriented Model Engine. This one is mainly based on a decision making algorithm in the perspective of an energy-aware instance allocation in a centralized brokerage Cloud environment. The algorithm provides a ranking of the offers (i.e., Data Sets as previously mentioned in the Paragraph 4.1.1) calculated on the basis of the inputs from the sustainability-oriented Model Simulator, and contemporary taking into account all the parameters reported in Table 1. The description of the algorithm is not the objective of this paper.

However, in order to introduce the experimental results thus to prove the goodness of the proposed approach, we briefly describe the main *formula* of the algorithm. It results in the *opt* value, which is an "optimum" index computed by the algorithm to weight each offer in terms of carbon dioxide emission (sustainability in gCO2) and cost ($-per-hour) as follows:

$$opt_{nProvider,nDataset} = A * (gCO2_{nProvider,nDataset}/gCO2_{worst} + B * (cost_{nProvider,nDataset}/cost_{worst}); \quad (2)$$

where $A + B = 1$.

In that Formula, $nProvider$ is an unique number which identifies the CSP' site, $Dataset$ is an unique number identifying the offer of that CSP, $gCO2$ quantifies the related CO2 emission to run an instance workload, *cost* is the service price in $-per-hour.

The A and B coefficients respectively represent the weight chosen for sustainability and for cost. Both CO2 emission and cost are normalized to the respective worst case calculated at each iteration on all the offers by all the CSPs.

5.4 Experimental Results

The resulting best offers of the J2CBROKER calculation are provided at each computation as ranking, both in the form of JSON documents and *on-screen*. In particular, each output reports three blocks, each one identifying a different ranking of offers based on the calculated *opt* values. The first one shows a ranking based on the best "green" offers (A=1;B=0); the second one results in a compromise between sustainability and cost (A=0.5;B=0.5); the last instead ranks offers ordered according to the service price (A=0;B=1).

Figure 2 allows the reader for a quick visual feedback about the confidence interval of the above-mentioned *opt* index to allocate a number N of 10 instances, with $A = 0.5$, $B = 0.5$, a number of 1000 samples and a 95% *confidence*. The *x-axis* reports the required *running time* in *hours (h)*. The values reported in Figure 2 are the result of a post-processing phase, by getting as input all the best *opt* values calculated at each run step. If we consider that for each run in our simulation, the worst case results in an *opt* index close to 1000, the energy-aware algorithm at Broker is able to select offers with an *opt* index very low when compared with the worst case. The result is a good compromise between sustainability and cost since it is as better as it is closer to zero.

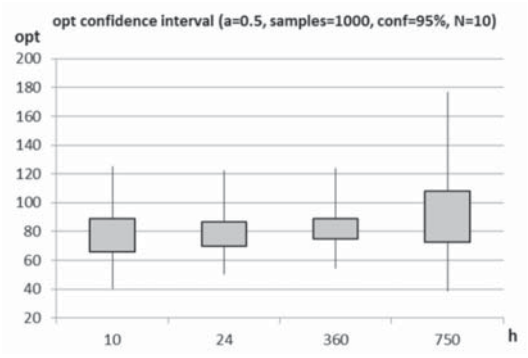

Figure 2: Confidence interval of the *opt* index for the allocation of ten instances.

6. CONCLUSIONS AND FUTURE WORK

The evaluation of performances in a real Cloud environment is too cost-and-time-consuming. This is why simulation tools can help researchers, Cloud Service Providers and customers to evaluate their proposals. In this paper we presented and discussed a multi-criteria Brokering Simulation Tool, we name *J2CBROKER*, able to simulate the discovering at a Cloud Broker of the most convenient offers delivered by Cloud Service Providers in cooperative Cloud ecosystems. By modeling different Cloud sites and the related economic offers on several criterion, we demonstrate how the proposed approach can accommodate different scenarios characterized by a different number of instances to allocate and based on both performance and business parameters. These last may also come from real datasets, and thanks to an "optimum" balance between them, it is possible analyze possible scenarios where a cooperative Cloud ecosystem can reduce the gap in competition with larger providers. Actually, the J2CBROKER is managed via command line. The results provided by the Brokerage Engine are available both through a JSON file and displayed on the screen at the user side.

In future works, we plan to realize an administrative dashboard which allows users to setup the simulation in a user-friendly manner. Furthermore we plan to develop a JAVA module to automatically show the results in a graphical way, by choosing between different representations.

7. ACKNOWLEDGMENTS

This work has been carried out in the framework of the CINI Smart Cities National Lab.

8. REFERENCES

[1] The Digital Single Market. https://ec.europa.eu/digital-single-market/en/digital-single-market.

[2] MarketsandMarkets. http://www.marketsandmarkets.com/.

[3] Standard ECMA-404. http://www.ecma-international.org/publications/standards/Ecma-404.htm.

[4] United Nations General Assembly (1987) Report of the World Commission on Environment and Development: Our Common Future. http://www.un-documents.net/wced-ocf.htm.

[5] The Ministry of Economy Trade and Industry (METI) Japan Project - Enhancing the Energy Efficiency and Use of Green Energy in Data Centers. http://home.jeita.or.jp/greenit-pc/sd/pdf/ds2.pdf.

[6] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE*, 14(13):1175–1220, 2002.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, Jan. 2011.

[8] M. Giacobbe, A. Celesti, M. Fazio, M. Villari, and A. Puliafito. A sustainable energy-aware resource management strategy for iot cloud federation. In *1st IEEE International Symposium on Systems Engineering, ISSE 2015 - Proceedings*, pages 170–175, 2015.

[9] M. Giacobbe and M. Villari. Tecniche avanzate di data management per la diagnostica avanzata in campo navale. In *SEA-MED Conference Proceedings*, pages 234–248. DIECII, 2014.

[10] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley. The java language specification, java se 8 edition, 2015.

[11] D. Kliazovich, P. Bouvry, and S. U. Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[12] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.*, 10(1):185–209, Mar. 2012.

[13] I. Patiniotakis, Y. Verginadis, and G. Mentzas. Pulsar: preference-based cloud service selection for cloud service brokers. *Journal of Internet Services and Applications*, 6(1):1–14, 2015.

[14] D. C. Plummer, B. J. Lheureux, M. Cantara, and T. Bova. Cloud services brokerage is dominated by three primary roles. *Gartner doc*, 2011.

[15] M. Radi. Efficient service broker policy for large-scale cloud environments. *CoRR*, abs/1503.03460, 2015.

[16] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, and E. Chang. Cloud service selection: State-of-the-art and future research directions. *Journal of Network and Computer Applications*, 45:134 – 150, 2014.

[17] S. Sundareswaran, A. Squicciarini, and D. Lin. A brokerage-based approach for cloud service selection. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 558–565, June 2012.

[18] B. Wickremasinghe, A. Prof, and R. B. Contents. Cloudanalyst: A cloudsim-based tool for modelling and analysis of large scale cloud computing environments, 2009.