Smart cloud federation simulations with CloudSim

Conference Paper · June 2013

DOI: 10.1145/2465823.2465828

3 authors:

Gaetano Anastasi
Ministero dell'Istruzione, dell'Università e della Ricerca
20 PUBLICATIONS **511** CITATIONS
SEE PROFILE

Emanuele Carlini
Italian National Research Council
75 PUBLICATIONS **764** CITATIONS
SEE PROFILE

Patrizio Dazzi
Italian National Research Council
130 PUBLICATIONS **1,194** CITATIONS
SEE PROFILE

Smart Cloud Federation Simulations with CloudSim

Gaetano F. Anastasi
Information Science and Technologies Institute
CNR, Pisa, Italy
g.anastasi@isti.cnr.it

Emanuele Carlini
Information Science and Technologies Institute
CNR, Pisa, Italy
emanuele.carlini@isti.cnr.it

Patrizio Dazzi
Information Science and Technologies Institute
CNR, Pisa, Italy
p.dazzi@isti.cnr.it

June 17, 2013

Abstract

In the last few years the broad diffusion of Cloud Computing has encouraged the proliferation of cloud computing providers. However, providers often propose their services using proprietary management software, interfaces and virtualization technologies. This strongly hinders the applications interoperability and migration across providers boundaries. Organizing providers in federations seems promising for addressing such issues, but it introduces other challenges to be faced, often requiring innovative approaches. Unfortunately, the evaluation of new solutions in a repeatable manner and under several configurations is a hard task to achieve using real Clouds platforms. For these reasons we propose SmartFed, a simulator for cloud federations that is able to model the richness typical of an environment with multiple cloud providers. We show the capability of SmartFed by simulating a sample mapping process for assigning applications to providers.

D.2.11Software EngineeringDomain-specific architectures
Design Cloud Computing, Cloud Federation, Simulator

1 Introduction

Cloud computing is an infrastructural paradigm that eliminates the need for maintaining and managing expensive computing hardware, and represents a concrete instantiation of the long-held dream of computing as a utility. Cloud computing allows the exploitation of resources as services according to a pay-per-use paradigm that relies on the existence of a Service Level Agreement (SLA) between the user and the provider, and suitable for expressing Quality of Service (QoS) terms that needs to be accounted, monitored and enforced. Currently, almost all IT behemoths offer their own cloud computing solution: Amazon[1], Google[2], Microsoft[3], to cite only the most popular. Most of the existing solutions come with their own management software and proprietary APIs, forcing users to operate according to specific communication protocols and virtualization technologies and, as a consequence, limiting the portability of the solutions. This phenomenon is known as *vendor lock-in* and strongly limits the migration of application and services across provider boundaries. For overcoming this issue, several standards – e.g. Open Cloud Computing Interface (OCCI) [2], Open Virtualization Format (OVF) [1] – have been recently proposed to foster the cooperation across different Cloud platforms and infrastructures. Unfortunately, these standards cover only specific aspects of the Cloud management stack, and none of them is yet universally adopted by cloud providers.

On the other hand, Clouds Federation [8] is one of the most promising approach both for abstracting the heterogeneity of different technologies adopted by different providers and for integrating multiple resources owned by different providers in a unified platform. Such approach has been adopted in Contrail[4], an ongoing EU project where we are conceiving and developing the *federation-support*, i.e. the components and the services supporting the cloud federation. In this paper, the terms federation and federation-support will be used interchangeably, by referring to the Contrail approach.

The federation role is not just to perform interface adaptation towards providers, instead it can be considered as a bridge linking cloud users and cloud providers, acting as a mediator between them. For instance, the federation-support manages user identities for accessing to different cloud providers and reports information about the usage of resources (i.e. accounting, billing). Also, users can submit applications and negotiate SLAs by exploiting at the same time and in a transparent manner all the federated providers.

By focusing on the application submission, the main task of the federation is to select the proper Cloud providers to run submitted applications. The input to the federation is composed by applications, i.e. a set of interconnected *appliances*, and SLAs providing the user requirements on a per-appliance basis. With the term appliance we identify a set of Virtual Machines (VMs) strictly cooperating to realize the fundamental block of an application (like a pool of

[1] http://aws.amazon.com/
[2] http://code.google.com/appengine/
[3] http://www.microsoft.com/windowsazure/
[4] http://www.contrail-project.eu/

web servers or the combination of firewall and database). The SLA associated to an appliance defines a set of requirements that must be addressed and enforced in order to properly execute the appliances. Upon the reception of the pairs application-SLA, the federation selects the proper Cloud providers to run the appliances according to the user requirements, the characteristics of the appliances and the related SLA. In order to effectively achieve this goal the federation exploits both static (e.g. cost-models) and dynamic information regarding the resources of Cloud providers.

Finding the best approach for resources selection and their allocation for running applications is a difficult task. Indeed, often it does not exist the "best" approach in general, but instead a most suitable approach given certain resources and applications. This implies the need of a careful design and a proper contextualization of either existing or new solutions to the considered environment. Besides the design, development and contextualization phase, there is the need of an evaluation aimed at testing different solutions on different target environment. Unfortunately, the evaluation can be both expensive and complex in a real federation of Clouds. A suitable solution can be represented by simulation tools, which offers the possibility of evaluating hypothesis prior to software development in a reproducible environment.

This paper presents SmartFed, a simulator specifically tailored for cloud federations built on top of CloudSim [7]. SmartFed employs a collection of software entities, which programmers can exploit to properly model a federation of Clouds. The whole framework has been designed for: (a) easing the testing of different Cloud configurations in repeatable and controllable environment, free of cost; (b) facilitating the finding of performance bottlenecks of the employed solutions before implementing them.

The main contribution of this paper is the presentation of our simulator and the associated framework. For this purpose, a federation of Clouds has been also modeled and presented. Moreover, we performed an initial analysis of the simulator by using a sample scheduling algorithm to allocate a predefined application on multiple datacenters.

The paper is organized as follows. In Section 2 we present the model of cloud applications and resources considered in SmartFed. Section 3 describes the architecture of SmartFed, including the relationship with CloudSim. A sample resource allocation strategy is presented in Section 4 for showing the viability of the proposed simulation solution. Finally, Section 5 presents the related work and Section 6 draws the conclusions.

2 Modeling Federations

In this section we present our proposal for modeling a cloud federation. For the sake of clarity, the discussion is divided in the following sections, which broadly represents the components of SmartFed:

- Application Model

- Resource Model
- Application Queue Model
- Resource Monitoring
- Resource Allocation Model

The first three components represent the main entities of the simulator: *what* to execute, *where* to execute, *when* the information needed to decide about the execution is available. The "Resource Monitoring" component aims to express the *quantity* and the *quality* of the information available for the allocation choices decision. The "Resource Allocation Model" component expresses *how* resources are associated with cloud applications, namely, it models the allocation strategies in the shape of mapping algorithms.

2.1 Application Model

Applications can be submitted to Clouds following very different approaches that are usually organized in three levels: *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS), *Infrastructure-as-a-Service* (IaaS). In fact, besides this common classification, each cloud provider can in principle support a different degree of expressiveness for the application description.

A promising proposal for a unifying standard description is represented by OVF, that describes applications at IaaS level using a configurable XML structure organized in different sections. Simplifying, OVF permits to describe applications as a hierarchical set of nodes, each one composed by a set of virtual machines inter-connected by virtual networks. Both machines and networks can be characterized by resources' requirements.

In order to provide an expressiveness similar to the one represented by OVF, in our model each cloud application submitted to the federation is represented by an undirected graph $\langle\ \mathbb{G}_N,\ \mathbb{E}_M\ \rangle$, where $\mathbb{G}$ represents the set of N vertices and $\mathbb{E}$ represents the set of M edges connecting the vertices. Each vertex $g_i \in \mathbb{G}$ embodies an *appliance*, that can be composed by multiple correlated VMs ($vm_{1..k} \in g_i$), each one potentially providing different services, e.g. one VM provides a firewall and another one provides a back-end database. Each edge $e_{i,j} \in \mathbb{E}$ represents an undirected communication path connecting vertices g_i and g_j.

Both vertices and edges are characterized by a set of *functional* (e.g. amount of available memory) and *non-functional* (e.g. reliability degree) requirements that must be addressed in order to run the application. In particular, for each $vm_i \in g_i$, in our model we limit the functional requirements req_i presented in Table 1. The first column reports the feature, whilst the second column indicates the condition for the requirement to be satisfied, with respect to the resource capacity. For instance, if the number of cores required by the VM image is equal or less to the number of core provided by the host machine then the requirement

Table 1: Functional requirements of a VM image

VM functional features	satisfied if:
CPU Type	Exact match
CPU Frequency	Equal or less
Number of cores	Equal or less
RAM size	Equal or less

is satisfied. As a consequence the requirements associated to a vertex g_i consist in the sum of the requirements of the VMs composing it.

Each vertex is also characterized by *non-functional* requirements sla_i that are expressed as a combination of SLA terms. In our model, we consider the SLA terms indicated in Table 2, where for each SLA term indicated in first column is also specified the satisfactory condition in the second column. For instance, if the `Resource Price` (the price a user accepts to pay for using that resource) is equal to 3$ per-hour then the SLA term is satisfied if the price made by that provider is equal or less to $3 per-hour.

Regarding the other terms of Table 2, the `Reliability Degree` is defined as the probability that a certain resource will be up and running in a defined portion of time t, e.g. a resource that has been working properly for 362 days of the last year has a year-reliability around the 99%. Finally, the `Elasticity Range` is defined as the range of instances associated to a VM during the application life-cycle, e.g. a VM that requires an elasticity range from 4 to 10 will be allowed to vary the number of its instances in the correspondent range of values.

Table 2: SLA terms

SLA terms	satisfied if:
Resource price	Equal or less
Reliability degree	Equal or greater
Elasticity range	In the range

An edge $e_{i,j}$ is an interconnection link connecting vertex g_i to vertex g_j. The edge $e_{i,i}$ with $i \in N$ can be also defined. In this case, it represents the characterization of the network connecting VMs of the same vertex. An edge is characterized by some non-functional requirements $\mathsf{sla}_{i,j}$, that are the same defined in Table 2 and some functional requirements $\mathsf{nreq}_{i,j}$ that are indicated in Table 3.

The first column of Table 3 reports the features, whilst the second column indicates the condition for the requirement to be satisfied, with respect to the link capacity. For instance, if the bandwidth required by the application link is equal or less to the bandwidth of the provider link, then the requirement is satisfied. Please note that the `Security capability` term indicates if and how the network link supports mechanisms for securing the connection, e.g. the RSA

Table 3: Network features

Network features	satisfied if:
Bandwidth	Equal or less
Latency	Equal or greater
Security capability	Exact match

algorithm.

2.2 Resource Model

As it happens for Applications, Cloud Resources can be seen at different levels of abstraction depending on the approach followed, e.g. the amount of spreadsheets that can be stored is a resource bound described at the level of SaaS, the number of rows that can be stored in a database is a typical PaaS level resource description, whereas the total size of a volume expressed in Terabytes is an information at the IaaS level.

In this work we concentrate on modeling resources at the IaaS level. Each IaaS provider is modeled with a datacenter c_i. We assume that each datacenter is composed by a set of hosts, that can run one or more virtual machines depending on their resources availability. The resources of each datacenter are interconnected by a network characterized by a specific set of features. Depending on its performance capabilities each datacenter can run a different number of applications, i.e. set of virtual machines. In our model we assume that the allocation units associated to a datacenter are defined in terms of virtual machines slots. The features of each slot as well as their number can vary datacenter by datacenter. More formally, a datacenter is defined as:

$$c_i = \{\mathsf{c\_feat}_i, \mathsf{c\_ava}_i, \mathsf{c\_nfeat}_{i,i}, \mathsf{c\_sla}_i\}$$

where $\mathsf{c\_feat}_i$ indicates the functional features (listed in Table 1) of the VM slots provided by the datacenter c_i. The $\mathsf{c\_ava}_i$ represents the number of VM slots for the datacenter c_i. As an example, a datacenter physically composed of 8 nodes having a quad-core i7 and 8 GBytes of RAM could provide 16 VM slots characterized by a dual-core i7 with 4 GBytes of RAM. The symbol $\mathsf{c\_nfeat}_{i,i}$ represents the functional features of the network interconnecting two vertices. If the two input indexes are the same, the functional features are related to the internal interconnection network of the datacenter. Such features are defined as in Table 3. Last, $\mathsf{c\_sla}_i$ is the non-functional features provided by the datacenter c_i. In this case, the considered features are only `Resource Price` and `Reliability Degree`, as the `Elasticity Range` term is meaningless for datacenters.

We assume datacenters to be connected to each others with internet-based network connections. Clearly, not all the network interconnection links provide the same features, e.g. bandwidth, latency, etc. Moreover, the actual usage of a link affects its performances. The usage of a link can vary either in a

predictable way or in a completely unforeseeable manner depending on several factors. In order to model this complexity, we introduced an *Internet Estimator* entity that estimates the properties of the network link interconnecting two datacenters. Such properties can be represented with the symbols $\mathsf{c\_nfeat}_{i,j}$ and $\mathsf{c\_sla}_{i,j}$, indicating the functional and non-functional requirements of the network link interconnecting datacenter c_i and datacenter c_j.

2.3 Application Queue Model

To be executed in one or more Clouds, each application needs to be associated with a set of proper resources. To find the most appropriated resources, the allocation algorithm shall consider the requirements of the application and evaluate the best assignments depending on the actual resources availability. Consequently, the time at which the application is submitted to the system is a relevant element for evaluating the ability of the allocation algorithm to map the applications to resources. Straightforwardly, an algorithm having a complete knowledge about all the application that will be submitted to the system can take better decisions than an algorithms that has only a partial view on the next applications that will be provided in input to the system.

The aim of the *Application Queue Model* is to define both the inter-arrival behavior followed by input applications and the information provided to the allocation algorithm regarding the application in input to the system.

2.4 Resource Monitoring

If the Application Queue Model defines the amount of information available to the allocation algorithm about the input applications, than similarly the monitoring subsystem defines the amount and the quality of information about the resources availability of the federated cloud providers. The "idea behind" is to have a way to model both the frequency at which the information is updated and the way it is aggregated, thus simplifying its management and reducing the amount of data to transfer from the providers to the federation. Our model allows to define the frequency of the updates and the behavior of data aggregators.

2.5 Resource Allocation Model

The Cloud Federation is associated to a set of providers and a set of applications. In this context, an allocation strategy defines a mapping between the resources belonging to federated Clouds and the applications submitted to the Federation.

In our model, we do not require that each application runs as a whole in the same datacenter, rather each appliance $g_i \in a$ can be executed by different datacenters. In other words, we do not consider the suitability of a Cloud to run an entire application but the ability to run an appliance, in isolation with respect to the other appliances. The analysis of appliances in isolation allows to reduce the complexity required to compute a suitable mapping that otherwise

would require, in the worst case, an exponential number of comparisons between the fractions of the application and the available resources. This reduces the amount of time for computing the allocation of resources to appliances, gaining in terms of scalability.

The mapping can be defined as a function $map(a) = Y$ that receives in input an application a and returns the matrix

$$Y = \begin{bmatrix} c_{1,1} & \cdots & c_{1,P} \\ \vdots & \ddots & \vdots \\ c_{N,1} & \cdots & c_{N,P} \end{bmatrix}$$

where the element $c_{i,j}$ represents the *j-th* datacenter suitable for running the *i-th* appliance of a. Thus, each row vector can be seen as the ranked list of datacenters (with $P \leq K$) suitable for running that appliance.

In order to compute this mapping, the federation needs information about the Cloud resources as well as about the network infrastructure interconnecting them. This information is composed by a "static" part that rarely changes (e.g. CPU type, installed memory) and a "dynamic" part that changes frequently (e.g. available memory, available network bandwidth). Static information can be gathered very often with low overheads and thus the probability of being up-to-date is very high. Dynamically changing information is instead gathered with an higher sampling time for not incurring in high overheads and thus it is available with some delay. One of the consequences is that a mapping computed by the federation could result valid on the basis of the information shared by the federation without being valid when actually enacted in the application deployment phase. A cloud federation simulator should provide the necessary abstractions to model these aspects.

3 Simulator Architecture

In this section we present the architecture of SmartFed. It extends and reuses several facilities of the CloudSim Simulator, which is described in more detail in the following. Then, Section 3.2 enters into the details of our implementation, discussing how all its components collaborate together to simulate cloud federations.

3.1 CloudSim Simulator

The CloudSim framework is one of the most known simulator for Cloud Computing environments. It is a discrete event simulator initially conceived in 2009 and it has been continuously extended with additional modules and features across the years, proven to be well-conceived for reusing and flexibility. For this main reason it has been chosen in this work for simulating the Contrail cloud federation model.

The CloudSim architecture is structured in three main layers. The bottom layer is the simulation engine, that supports the core functionalities of the simulation, such as the queuing and processing of events, the creation of software entities and the clock simulation management. The CloudSim layer, that stands in the middle, provides support for the modeling and the management of cloud datacenters, allowing the simulation of VM provisioning, application execution management and providers monitoring. The User Code layer is the top-most one and provides support for high-level modeling of application workloads and Cloud resources. Moreover, it provides tenants for implementing scheduling policies through the broker entities.

For the purpose of modeling the Contrail Federation, the `Broker` is the most interesting one to extend. In fact, the broker's task is to assign virtual machines to cloud providers, referred to as `Datacenter` in CloudSim. This task is one of the key roles of a Federation, along with all the decisions that involves the run-time support, such as elasticity or live migration. These datacenters are taken at federation level and should be implemented at the top-most level of CloudSim.

The relevant point for simulating Cloud Federations is a proper representation of applications. Plain CloudSim essentially represents all the input applications as entities (referred to as `CloudLet`) that are independent one from each others. This hampers the task of properly representing the relationships among VMs composing complex Cloud applications, which happen to have peculiar requirements related to links interconnecting VMs that can also be hierarchically organized, as we described in Section 2.

These are the main differences that distinguish the Contrail cloud federation model from the federation model embedded in CloudSim. In fact, federated cloud datacenters are seen as peers in CloudSim and thus they can cooperate for implementing policies like compensation or inter-cloud load balancing. It is clear that such model does not fit the Contrail one, as each Contrail Cloud is an autonomous system that competes with the others belonging to the same Cloud Federation.

3.2 Federation Simulator

This section describes the entities composing SmartFed (see Figure 1). The entities are reported by specifying their role and the programming model that should be followed to implement peculiar aspects of cloud federations. The rest of this section is organized in five subsections devoted to describing the entities

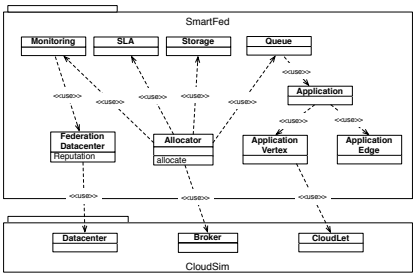

Figure 1: Main entities of SmartFed

supporting, respectively, the implementation of Cloud Applications, the Application Queue, the Internet Estimator, the Monitoring Hub and the Allocator.

3.2.1 Application

Each cloud application is modeled as a graph, according to the model described in Section 2. Each vertex is represented by an ApplicationVertex (AV) entity consisting in a set of Cloudlets (CLs). Each AV is also associated with a set of homogeneous VM, one for each Cloudlet. Each edge of the graph is represented by an ApplicationEdge (AE) entity that defines the requirements associated with the communication link interconnecting the CLs, more in detail in our model we consider bandwidth, latency and channel security requirements. As an example, let us consider an application where $\{CL_1, CL_2\} \in AV_1$ and $\{CL_3\} \in AV_2$, where AV_1 and AV_2 are connected through AE_1. This indicates that the application requires CL_1 and CL_2 to be connected with CL_3 with a link satisfying the characteristics defined in AE_1.

3.2.2 Application Queue

In SmartFed, the arrival of the applications into the Federation can be properly defined and tuned. In particular, this can be done exploiting the Application Queue entity. It provides the support for defining the inter-arrival of applications into the Federation. From an operative point of view, the programmer has to specify the arrival time of each application in the system either choosing a standard distribution among the ones available in the simulator (e.g. the $\alpha-$distribution), or to directly provide the logic for a custom one. This last possibility can be exploited by simply implementing a method that takes in input all the Applications to consider during the simulation and returns back for each of them a timestamp describing the arrival time of the Application in the Federation.

3.2.3 Internet Estimator

As we anticipated in Section 2, SmartFed provides a support for modeling the network bandwidth across datacenters. The Internet Estimator provides this support by describing the links between federation datacenters, associating for each link the corresponding latency, bandwidth and security mechanisms.

The Internet Estimator derives from the CloudSim's NetworkTopology, but it adds the ability of pre-reservation of links during the mapping phase. Basically, it provides the primitives for: (i) defining the links inter-connecting couples of datacenters, (ii) getting information about links, and (iii) reserving and releasing bandwidth. It is worth to point out that our Internet Estimator assumes that the link connecting two datacenters is not influenced by other allocations not explicitly involving that link, i.e. we assume that each link is dedicated.

3.2.4 Monitoring Hub

The ability of modeling the amount and the quality of the information available for the federation about datacenters' resources is particularly useful to model this kind of systems. Cloud federations are expected to deal with notable amounts of highly distributed resources, organized in several different datacenters. It is not realistic to assume the federation to have all the up-to-date information about the actual usage of cloud providers. In fact, in order to model realistic simulations it is fundamental to have the possibility to specify the frequency (w.r.t. the time units) of polling requests issued by federation to retrieve information about cloud providers. Moreover, it is also useful a support that allows to specify the amount of past usage information about datacenters at federation side. The SmartFed extension allows programmers to characterize this information by customizing the Monitoring Hub entity, that is devoted to this end.

3.2.5 Allocator

The central entity of our CloudSim extension is the Allocator. By customizing it, the programmer can implement the allocation algorithm, i.e. the functional code that determines the associations between applications and datacenters. In literature there exist several different algorithms that have been proposed to this end. Almost all of them consider the features associated to applications and the datacenters (and their resources). Finer solutions also consider the networks interconnecting datacenters to optimize the allocations also considering the communications among the components of an application (in our case expressed by the edge of the graph). Some solutions also consider long-term information about datacenters, e.g. their workload or their reputation.

All these information are available to programmers exploiting our CloudSim extension. Indeed, the Allocator entity can access the Application Queue, the Internet Estimator and the Monitoring Hub. It also provide a persistent storage in which can be stored long-term information about datacenters.

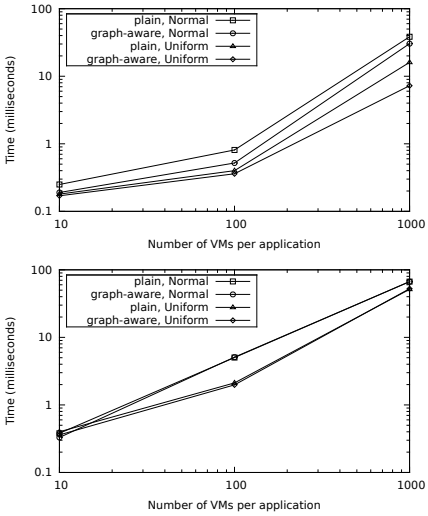

Figure 2: Mapping time obtained with $D = 10$ (left) and $D = 100$ (right)

4 Hands on SmartFed

The purpose of this section is to present some preliminary results obtained with the proposed simulator. We generate different simulation scenarios differentiated by the following elements: (i) number of CLs for each application; (ii) number of datacenters; (iii) distribution of hosts in datacenters; (iv) resource allocation algorithm.

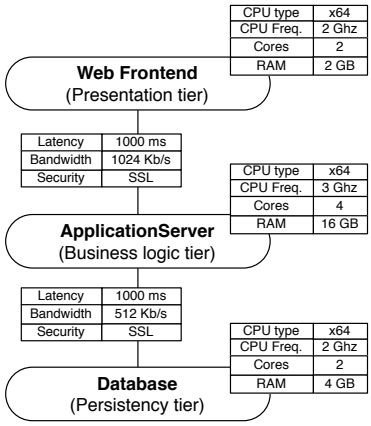

Figure 3: The 3-tier application used in the simulation

We consider a 3-tier web application composed by: (i) a web front-end tier for accessing the application; (ii) an application tier running the business logic; (iii)

a persistence tier handling the storage. Such application, depicted in Figure 3, is modeled as a graph $\langle\ \mathbb{G}_3,\ \mathbb{E}_2\ \rangle$ (see Section 2.1).

Each g_i is composed by a set of homogeneous VMs with the same hardware requirements, as defined in Table 1, whereas each $e_{i,j}$ has associated the set of network requirements defined in Table 3. The particular values used in these experiments are also depicted in Figure 3. For the sake of simplicity, we do not consider SLA terms in this use case.

First, the number of CLs is varied for the entire application, taking the following values: 10, 100 and 1000. Without loss of generality, we assume that each CL runs on a single VM, thus the two entities correspond one each other. The number of VMs per tier is assigned so that the web front-end and the database have each 20% of the VMs, and the application server has the remaining 60%.

Second, the federation manages a set of D datacenters each one corresponding to a cloud (DCs), that can be seen as collections of hosts. In these experiments, D assumes the following values: 10, 100. The total number of hosts (H) assigned to DCs is roughly a 4x the number of VMs in the application.

Third, the number of hosts in a DC may impact on the mapping algorithm. As an example, consider the case in which the first datacenter has a number of hosts that is sufficiently large for containing all the VMs. In this case no optimization can be enacted for a greedy approach, as all the variants of this algorithm will map the application on that datacenter. For this reason, we employ two strategies for assigning hosts to DCs. In the first strategy, DCs have an uniform number of hosts, i.e. every DC has $\lceil \frac{H}{D} \rceil$ hosts. In the other strategy we distributes hosts according to a normal distribution $\mathcal{N}(m, \sigma^2)$, with $m = D/2$ and $\sigma^2 = D/4$, forcing DCs to have at least one host.

Fourth, the algorithm used for testing the simulator follows a greedy strategy but we implemented two versions of it, denoted as *plain* and *graph-aware*. In the plain version, a DC is selected by checking whether it can run the VM and if the network requirement can be met. This process is repeated for each VM, regardless of the previous assignments. Such approach is the one commonly exploited by CloudSim. On the other hand, the graph-aware version exploits the application as a graph. Indeed, it tries to allocate VMs belonging to the same type (and hence, the same tier of the application) in a DC. If the DC cannot manage all the VMs, the algorithm tries to place all the remaining VM in an another DC, and so on until the tier is not fully allocated. The algorithm terminates successfully when all the VMs are assigned and running.

Some simulations have been run on a machine equipped with Java 7, 16GB of RAM, an Intel i5-2550 quad core @3,30 Ghz and the mapping times obtained with SmartFed have been measured. Figure 2 shows (in log-scale) the mapping time as a function of the number of VMs. In particular, the left graph of Figure 2 shows results obtained when $D = 10$, whilst the right graph shows results obtained when $D = 100$. Results are plotted for the combinations of the discussed configurations regarding the mapping algorithm and the distribution of hosts. It can be noticed that the mapping time slightly increases with more DCs but results are of the same magnitude order.

In general, the graph-aware algorithm obtains lower mapping times with respect to the plain one, especially with $D = 10$. In fact, the graph-aware version tries, once that a suitable DC has been found, to map all the VMs of a vertex in the same DC, avoiding to tries on unfitting DCs that may be in the beginning of the ranked list. Figure 2 also highlights that the distribution strategy of hosts to DCs affects the mapping time. By using the uniform distribution, each DC have a fair amounts of hosts, which prevents them to be filled soon. On the other side, by using the normal distribution, only few DCs have an high amount of hosts, and the other DCs are filled very early.

Such results show that SmartFed can easily support simulation and confrontation of different algorithms and configurations in the context of cloud federations.

5 Related Work

This section briefly analyzes related work in the fields realated to this paper, being cloud federation, resource allocation and cloud simulation.

As already discussed in Section 1, our approach to cloud federation is the Contrail one. Other works exist that aim at integrating multiple and heterogeneous clouds. For instance, InterCloud [6] is a federated cloud computing environment that aims at provisioning application in a scalable computing environment, achieving QoS under variable workload, resource and network conditions. This work exploits a central coordinator that exposes basic functionalities for resource management such as scheduling, allocation, workload and performance models. In the Reservoir project [13], the authors propose an architecture for an open federated cloud computing platform. In such architecture, each resource provider is an autonomous entity with its own business goals. A provider has the ability to choose the other providers to federate with. In the work by Celesti et al. [9], the *Dynamic Cloud Collaboration* is proposed, an approach for setting up highly dynamic cloud federations. In order to dynamically federate a new provider, a distributed agreement must be reach among the already federate partners. Conversely from these approaches, Contrail aims at adopting a more cloud-independent approach, where the federation component plays a central role, incorporating functionalities such as SLA management and advanced mapping.

The problem of allocating physical resources to applications has been tackled many times in the past, especially in the Grid and service-oriented communities [3, 11, 5, 12, 10]. In the work by Blythe et al. [5] two families of resource allocation algorithms are identified: task-based algorithms, that greedily allocate tasks to resources, and workflow-based algorithms, that search for an efficient allocation for the entire workflow. Mika et al. [12] propose an application model similar to the one proposed in this paper. Indeed, they consider computing and networking resources in the grid and workflows represented as a directed acyclic graph (DAG) of communicating tasks, with associated computing and networking requirements, similarly to what we do.

With respect to the Grid, in the Clouds an additional issue must be considered. In fact, services are encapsulated in VMs and thus scheduling services may comprise finding enough resources for a VM to be deployed. As an example, the work by Wang et al. [14] addresses the problem of scheduling parallel tasks within a SOA by taking into account multiple resources (software, CPU, memory) required by the corresponding VMs and each service is scheduled when all the required resources are available.

As we already discussed in the paper, the simulation of a cloud is an indispensable tool to test scheduling and mapping algorithm. In this work we have exploited CloudSim [7], since it has proved to be efficient and flexible enough to support heterogeneous systems and evaluations. In fact, some works exist that have proficiently used this simulator. For instance, the work by Wu et al. [15] discusses an SLA-driven mapping on Software as a Service platform, measuring through CloudSim the performances in terms of SLA violations and

economical cost. Also the work by Beloglazov and Buya [4] uses CloudSim for measuring performances of an energy-aware resource management algorithm for Infrastructure as a Service platforms.

6 Conclusion

The exploitation of cloud federations, like the one proposed by the Contrail platform, requires a deep analysis in term of correctness and performances. Considering the large scale of the cloud infrastructure and applications, simulations help as a tool for evaluating the feasibility of a solution. In this context, this paper proposes a model of the application and cloud resources targeting the Contrail platforms. These concepts were analysed with simulations performed by exploiting the CloudSim simulator. To this end, we developed additional modules supporting a richer model of SLA both in applications and in the cloud resources, and additional mechanisms to support off-line mapping of applications before their actual deployment. The paper proposed an evaluation of the scalability of application mapping in a simulated contrail federation, demonstrating the feasibility of the approach. Other issues, such as user authentication and authorization, are left as future work.

7 Acknowledgments

The authors acknowledge the support of Project FP7- 257438, Contrail: Open Computing Infrastructures for Elastic Services (2010- 2013).

References

[1] Open Virtualization Format Specification, Version 1.1. Specification, DMTF, Jan. 2010.

[2] Open Cloud Computing Interface - Core. Specification, Open Grid Forum, June 2011.

[3] D. Batista and N. da Fonseca. A brief survey on resource allocation in service oriented grids. In *Globecom Workshops, 2007 IEEE*, pages 1–5, nov. 2007.

[4] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 2011.

[5] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759 – 767 Vol. 2, may 2005.

[6] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. *Algorithms and Architectures for Parallel Processing*, 6081/2010(LNCS 6081):20, 2010.

[7] R. 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. *Software: Practice and Experience*, 41(1):23–50, 2011.

[8] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti. Cloud federations in contrail. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7155 of *Lecture Notes in Computer Science*, pages 159–168. Springer Berlin Heidelberg, 2012.

[9] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to enhance cloud architectures to enable cross-federation. In *3rd International Conference on Cloud Computing*, pages 337–345. IEEE, 2010.

[10] T. Cucinotta and G. F. Anastasi. A heuristic for optimum allocation of real-time service workflows. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–4, dec. 2011.

[11] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[12] M. Mika, G. Waligora, and J. Weglarz. Modelling and solving grid resource allocation problem with network resources for workflow applications. *Journal of Scheduling*, 14:291–306, 2011.

[13] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, and J. Caceres. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4, 2010.

[14] L. Wang, G. von Laszewski, M. Kunze, and J. Tao. Schedule distributed virtual machines in a service oriented environment. In *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, AINA '10, pages 230–236, Washington, DC, USA, 2010. IEEE Computer Society.

[15] L. Wu, S. Garg, and R. Buyya. SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204. IEEE, 2011.