Optimal Distribution of Applications in the Cloud

Vasilios Andrikopoulos, Santiago Gómez Sáez,
Frank Leymann, and Johannes Wettinger

IAAS, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
{andrikopoulos,gomez-saez,leymann,wettinger}@iaas.uni-stuttgart.de

Abstract. The emergence of the cloud computing paradigm introduces a number of challenges and opportunities to application and system developers. The multiplication and proliferation of available offerings by cloud service providers, for example, makes the selection of an appropriate solution complex and inefficient. On the other hand, this availability of offerings creates additional possibilities in the way that applications can be engineered or re-engineered to take advantage of e.g. the elastic nature, or the pay per use model of cloud computing. This work proposes a formal framework which allows to explore the possibility space of optimally distributing application components across cloud offerings in an efficient and flexible manner. The proposed approach introduces a set of innovative in their use concepts and demonstrates how this framework can be used in practice by means of a running scenario.

Keywords: application topology, distribution optimization, cloud computing, operational expenses.

1 Introduction

The cloud computing paradigm offers a well documented set of benefits to enterprises and individuals with respect to transferring capital to operational expenses, potentially unlimited access to computational resources, and utility-based charging for the use of these resources [4]. In this respect, cloud computing offers a platform for innovative information systems that are partially or completely implemented using cloud offerings. In order however to reap the full benefits of cloud computing, application design and development must move beyond the mere re-packaging of applications in virtual machines (VMs) and offering them as part of Infrastructure as a Service (IaaS) solutions [1]. For example, novel services like Database as a Service (DBaaS) offerings can be used in designing and realizing new applications, or while migrating and accordingly adapting existing applications for the cloud. Furthermore, when considering the variation of pricing models across cloud providers [22], it becomes possible to select from different cloud offerings, e.g. different configurations of the AWS

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 75–90, 2014.

EC2 service[1], in order to identify an optimal in terms of operational expenses distribution of the application.

Toward this goal, a number of approaches provide decision support for migrating existing applications to the cloud, see for example [2,10,16]. However, these approaches do not consider as part of their process the *application topology*, i.e. the combination of *application-specific components*, *middleware solutions* like the application server used, and the underlying *infrastructure* (VMs on either a local server, or on cloud offerings) hosting both of them and allowing the application to operate. Using the taxonomy proposed in [1], such approaches usually provide support for migration type III, meaning that the whole *application stack* (components, middleware and OS) is bundled in a VM image and moved to a cloud provider for hosting. In this respect, these approaches are limited in their capabilities when considering the distribution of the application across cloud offerings and/or local, in-house servers.

On the other hand, initiatives like the TOSCA standard [6], Cloud Blueprints [20] or CloudML [7] allow for a portable and interoperable topological description of the application stack that can be used for the distributed deployment of the application across cloud providers. Using these initiatives, it becomes possible for the application developer to explore the application design space and model which cloud offering to use to host which parts of the application stack. However, what these approaches lack is decision support capabilities towards *optimally* selecting the best of the identified application topologies in a given situation. This is a deficiency that this work aims to address by bringing together cloud migration decision support with these cloud-aware topology description approaches. The proposed approach does not make any assumptions with respect to the technologies used and as such it is suitable for use in both generic and domain-specific information systems.

The main contribution of this work can therefore be summarized as a technology-agnostic formal framework that provides the means to:

- Model, verify and automatically generate alternative scenarios for the distribution of an application stack across cloud offerings. Applications in this context may entail a complete information system, or only part of it.
- Evaluate each one of these distribution scenarios with respect to various dimensions using different criteria, and allow the selection of an optimal scenario given the application needs.

The remaining of this paper is structured as follows: the following section (Section 2) discusses a motivating scenario that illustrates the challenges that this work is addressing. Section 3 builds on existing models and languages to provide a formalization of the notion of application topology and affiliated concepts. Section 4 uses this formalization to develop a method for the optimal selection between alternative (acceptable) application topologies, which is demonstrated in practice in Section 5. Related approaches are discussed in Section 6, and the paper concludes in Section 7 by providing also the outline for future work.

[1] Amazon Web Services (AWS) EC2: `http://aws.amazon.com/ec2/`

2 Motivating Scenario

For purposes of further motivating this work we adapt the Web Shop application topology discussed in [6]. We abstract away from the TOSCA notation used in [6] and represent the topology of the application as the nodes and edges with solid lines in the graph of Fig. 1. The application itself consists of three tiers: front end, back end and persistence as a database. The WebShop_Frontend component is developed as a set of PHP files deployed in a PHP container. The container is in turn configured and deployed as an Apache Web server module, with the server running inside a Windows 2003 Server OS installed on top of an IBM zSeries server. The WebShop_Backend component is a Web application packaged in a WAR (Web application ARchive) file running inside an Apache Tomcat servlet container (requiring also the installation of the Oracle Java Virtual Machine (JVM) in the same OS), while the MySQL RDBMS is used as a database server for the ProductDB database. The latter two tiers are deployed in a Windows 7 image provided by the Amazon Web Services (AWS) EC2 service as part of their Reserved Instances offerings[2].

The application topology in Fig. 1 is already deployed in a distributed manner, in the sense that the components from the different tiers are deployed and operated in different infrastructure solutions, with the front end in a physical server on premises, and the back end and persistence in an IaaS offering. However, the application topology shown in Fig. 1 is only one of the possibilities for distributing the application. As also shown in Fig. 1, and marked with dashed lines in the figure, it is also possible to separate the back end from the persistence tier and deploy them in different EC2 offerings (denoted by the 'alt_hosted_on' relationship), one or both of which could be an Ubuntu Linux OS image. Furthermore, the ProductDB database could also be migrated to the Amazon RDS[3] DBaaS solution, which is compatible with the MySQL RDBMS.

Each one of these topologies has a different impact on essential characteristics of the application such as operational expenses, deployment time, scalability opportunities, performance, etc. Different pricing models are used, for example, for IaaS and DBaaS offerings, taking into account different parameters, e.g. number of CPUs per VM in the former case and size of egress traffic per month in the latter. Furthermore, migrating the ProductDB to AWS RDS can be better suited for profiting from some characteristics offered out of the box by DBaaS offerings, e.g. multi-instance management, high availability, automated scaling, etc., as re-engineering the application to deal with data consistency issues across database replicas is not required.

There are therefore two major challenges that this work is addressing: first, *how to infer the existence of possible topologies* for a given application, and second, *how to optimally select amongst these alternative topologies* for a given set of characteristics like operational costs. In the following section we introduce a formal framework that provides us with the fundamentals necessary towards dealing with these challenges.

[2] AWS EC2 Instance Types: `http://aws.amazon.com/ec2/instance-types/`
[3] AWS Relational Database Service: `http://aws.amazon.com/rds/`

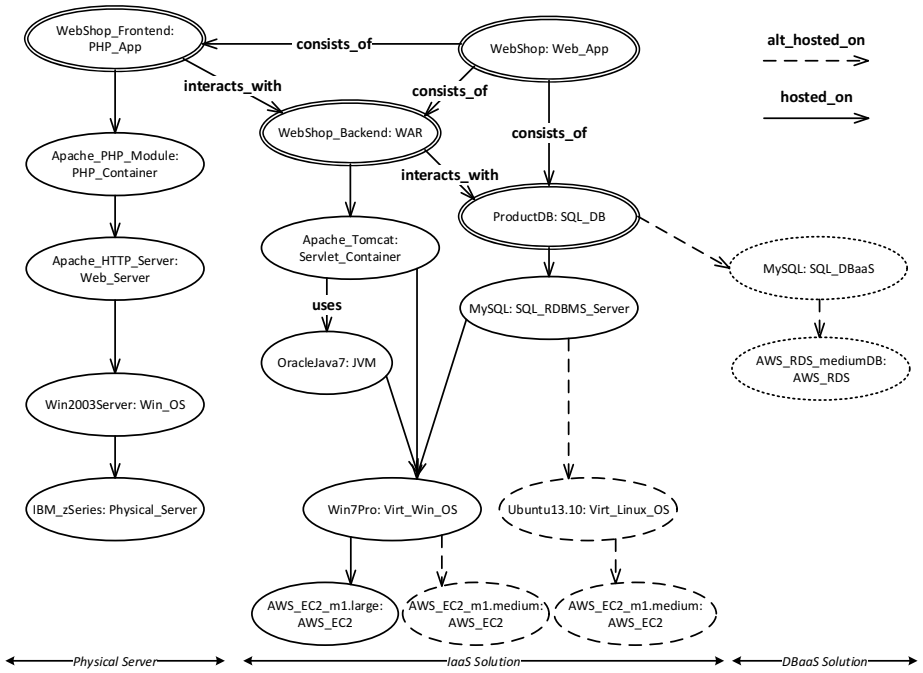

Fig. 1. The Web Shop Application Topology (adapted from [6])

3 Application Topology Fundamentals

So far we have been using the term 'application topology' in a rather informal manner to denote the model for the deployment of the application components in middleware solutions (e.g. the Apache Tomcat container in Fig. 1), and the consequent deployment of the resulting software stack in an appropriate infrastructure solution (e.g. the zSeries server, or the EC2 VM offering). Before proceeding further we first formalize this notion:

Definition 1 (Application Topology). *An* application topology *is a labeled graph* $G = (N^L, E^L, s, t)$ *where* N *is a set of nodes,* E *is a set of edges,* L *a set of labels, and* s, t *the source and target functions* $s, t : E^L \rightarrow N^L$*. The topology graph is called* typed*, if the label set* L *contains only elements* <name:type> *(for nodes) and* <type> *(for edges), in which case the graph is denoted by* T*.*

Most existing works for cloud-oriented topology description like the TOSCA specification [6], the Cloud Blueprinting approach [20], and the CloudML language [7], or involving such a description as in, e.g. the MOCCA [13] framework, use this typed topology graph model, in order to provide a concrete description of the application and middleware components and cloud offerings involved under a unified model. Similar approaches are also used by cloud service providers like

Amazon with CloudFormation[4], as well as the OpenNebula initiative[5] or OpenStack Heat[6], with a clear orientation towards facilitating and/or automating the deployment, provisioning and management of applications on cloud solutions. For this purpose they need a complete and a priori defined description of the application topology that can be distributed across multiple cloud offerings.

However, as discussed in the previous section, this is a limited view of the possibilities available in distributing the application across cloud solutions. Looking at the case of the Web Shop application, it can be observed that there is a conceptual distinction between the application components on one hand (denoted with double lines in Fig. 1), and the middleware components like the Apache Web server and the cloud offerings like the AWS EC2 service on the other. More specifically, while the former part is unique and specific for the Web Shop application, the latter can actually be reused and even shared across multiple applications similar to the Web Shop. In this respect, the typed topology model used by approaches like TOSCA should therefore only be interpreted as *one possible instantiation* of the application topology. In order to be able to model and explore this possibility space, the notion of a *type graph with inheritance* as formally defined in [5] and [12] can be used:

Definition 2 (Type Graph with Inheritance, following [5]). *A* type graph with inheritance TG_I *is a triple* (TG, I, A) *consisting of a type graph* $TG = (N, E, s, t)$ *(with a set of nodes* N*, a set of edges* E *and a target function* $s, t : E \to N$*), an inheritance graph* I *sharing the same set of nodes* N*, and a set* $N^A \subseteq N$*, called abstract nodes. For each node* $n \in I$ *the* inheritance clan *relation is defined by* $clan(n)_I = \{n' \in N \mid \exists \,\text{path}\ n' \xrightarrow{*} n \in I\}$ *where* $n \in clan(n)_I$ *(i.e. the* path *of length 0 is included).*

TG_I is therefore a graph where the nodes and edges are types, and where edges denoting the inheritance/subtype relation type, as in UML class diagrams, is allowed between nodes. Bardohl et al. use the concept of *abstract* nodes in [5] for types that have only inheritance relations with other nodes, meant to denote generic classes of nodes like e.g. Web Server. Using the clan morphism relation $clan(n)_I$ allows for navigating the inheritance-type edges in TG_I graphs, which is instrumental in producing typed graphs. In this respect, thinking of the application topological description as a graph morphism over TG_I produces potentially multiple typed topology graphs depending on the availability of sibling nodes in inheritance relations with abstract nodes (e.g. 'Apache HTTP Server' and 'IBM WebSphere' for the 'Web Server' node). The concept of *viable topology* builds on this capability:

Definition 3 (Viable Topology). *A typed topology* T *is* viable *w.r.t. a type graph with inheritance* TG_I*, iff all elements of* T *are labeled (typed) over the elements of* TG_I*, i.e. there exists a graph morphism* $m : TG_I \to T$ *which uses the inheritance clan relation.*

[4] AWS CloudFormation: `http://aws.amazon.com/cloudformation/`
[5] OpenNebula.org: `http://opennebula.org/`
[6] OpenStack Heat: `https://wiki.openstack.org/wiki/Heat`

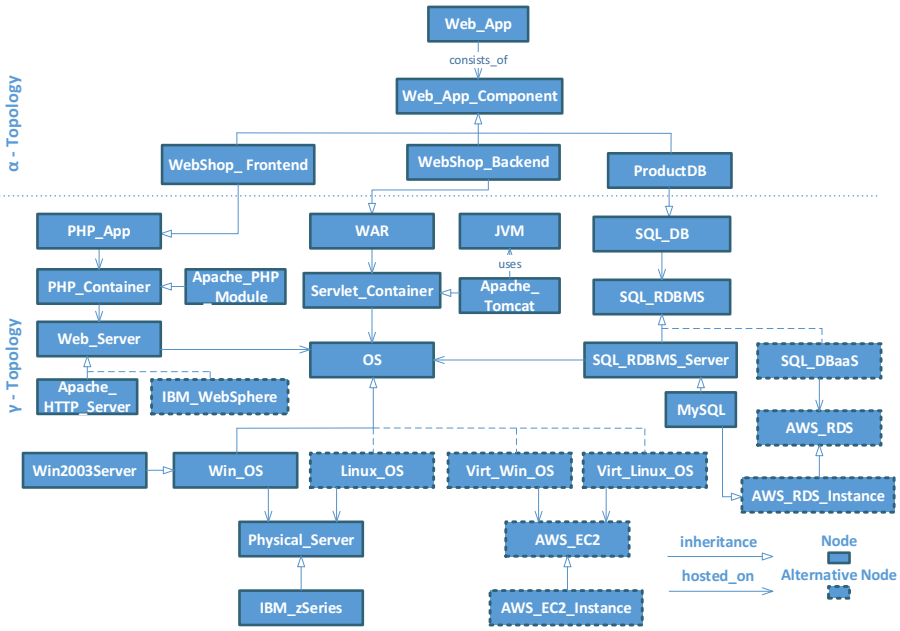

Fig. 2. The (Extended) μ-topology of the Web Shop Application

Based on this definition, the topology of the Web Shop application of Fig. 1 can therefore be classified as viable under the TG_I graph of Fig. 2. In addition to including the same types as the typed topology of Fig. 1 connected through edge types inheritance, 'consists_of' and 'hosted_on', the TG_I also incorporates types like 'Linux_OS' as a subtype of the 'OS' node that were not included in Fig. 1 (marked with dashed lines in Fig. 2). There are two ways to look at the morphism m that translates TG_I to T: *top-down*, with T being generated or validated against TG_I, and *bottom-up*, with TG_I being abstracted from one particular typed topology T and potentially being reused across different viable topologies. In order to facilitate the discussion, the following terms are being introduced:

Definition 4 (μ, α and γ-topology). *The type graph with inheritance TG_I for a viable application topology T is called its* μ-topology. *We denote by* α-topology *the application-specific sub-graph of a μ-topology, and by* γ-topology *the non application-specific (and therefore reusable) sub-graph of a μ-topology.*

In the μ-topology of Fig. 2, for example, the upper nodes (above the dotted line) belong to the α-topology of the Web Shop application, while the lower nodes (below the line) belong to its γ-topology. The distinction between α- and γ-topology is purely functional in nature, and the border between them can be moved dynamically per application to accommodate the application needs. For

example, if the Web Shop front-end component requires exclusively a Windows 2003 Server OS due to the way that it was implemented, then the whole subgraph under it can be moved to the α-topology of the application to reflect this fact. Alternatively, all the necessary components for the front end (PHP container, Web server and OS) can be bundled together with the Web Shop front end component, in which case the 'WebShop_Frontend' node in the graph of Fig. 2 can be replaced by an equivalent 'WebShop_FrontendBundle' (in the α-topology) that is connected directly with the 'Physical_Server' node with a 'hosted_on' relation. In this manner, resource requirements as explicit constraints on the possible topologies, as discussed in both [6] and [20], can be specified.

Finally, a set of viable topologies $\mathcal{V}$ for an application can be generated given the α-topology of the application and a generic γ-topology that can even be standardized in a domain or enterprise by merging the two graphs using the inheritance relationship. Using the resulting μ-topology, a set of viable topologies can then be inferred from the μ-topology by applying different morphisms $m^{(i)} : TG_I \rightarrow T^{(i)}, i \geq 1$ to it, resulting in different topologies $T^{(i)} \in \mathcal{V}$. We assume without loss of generality that there always exists a viable topology for an application, i.e. $|\mathcal{V}| \geq 1$. In the following sections we only consider viable topologies in the discussion, unless explicitly stated otherwise.

4 μ-Topologies and Distribution Optimization

The introduction of μ-topologies provides us the tools to deal with the first of the challenges identified in Section 2, i.e. inferring the existence of possible (viable) topologies for an application. The richer in terms of available types the γ-topology used is, the bigger the size of the viable topologies set $\mathcal{V}$ for the application. In the following we build on the introduced formalisms in order to address the second identified challenge, i.e. optimal w.r.t. a given set of dimensions selection among these possible topologies for a given set of parameters, in a formal manner.

For this purpose, we first introduce (a set of) utility functions as the means to quantitatively evaluate a topology along one or more dimensions, and then we formulate the optimal topology selection problem in order to identify the steps involved in solving it.

4.1 Optimization Utility Function

Let's assume a set of functions $\mathbb{F}$ on the domain of real numbers $\mathbb{R}$ of the form $\mathbb{F} = \{f(a_1, \ldots, a_n) \,|\, n \geq 1, f : \mathbb{R}^* \rightarrow \mathbb{R}\}$, and a mapping function f_{map} from each topology in the set of *all* viable topologies for all applications $\mathbb{V}$ to this set $f_{map} : \mathbb{V} \rightarrow \mathbb{F}$. We then denote by $u^{(i)} \in \mathbb{F}$ the function $u^{(i)}(a_1, \ldots, a_n) = f_{map}(T^{(i)})$ and by $A^{(i)}$ the set of arguments $\{a_1, \ldots, a_n\}$ of $u^{(i)}$; as a shorthand for $u^{(i)}(a_1, \ldots, a_n)$ we equivalently use $u(T^{(i)})$ in the rest of this document. Providing concrete values $P^{(i)} = (p_1, \ldots, p_n)$ for the arguments $A^{(i)}$ allows for the *evaluation* of the function, i.e. $eval(u(T^{(i)}), (p_1, \ldots, p_n)) = u^{(i)}(p_1, \ldots, p_n)$.

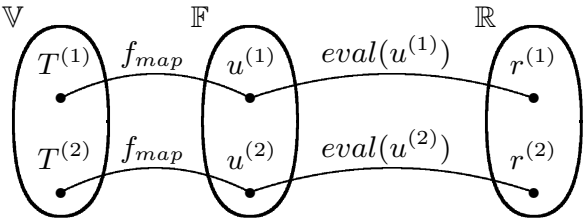

Fig. 3. Schematic Representation of the Sets $\mathbb{V}, \mathbb{F}, \mathbb{R}$ and the Mappings Between Them

Function $u^{(i)}$ is essentially the *utility function* for the topology $T^{(i)}$, in the sense that it evaluates the dimension on which the optimization takes place by providing a mapping from the set $\mathbb{V}$ to $\mathbb{R}$ as shown in Fig. 3, where $r^{(i)} = eval(f_{map}(T^{(i)}), (p_1, \ldots, p_n)) = eval(u^{(i)}(a_1, \ldots, a_n), (p_1, \ldots, p_n)), r^{(i)} \in \mathbb{R}$. For purposes of simplifying the discussion we assume that functions $u^{(i)}$ are by definition monotonic.

Different utility functions can be defined for the same topology depending on which dimension is taken under consideration. Furthermore, the concrete definition of each utility function depends on the types of the nodes in the μ-topology of the application. For example, for the initial topology $T^{(1)}$ of the Web Shop application in Fig. 1 the operational expenses of the application are decomposed into the expenses $opex_{zSeries}$ of operating the IBM zSeries server hosting the Web Shop front end for a time period of τ months, and into the cost $opex_{EC2\_m1.large}$ of using the AWS 'EC2 m1.large' offering for the same time period. For the former, $opex_{zSeries}$ can be calculated using a method like the one discussed by Walker in [23] as the product of the electricity cost, power consumption for operation and cooling, server utilization and number of hours of operation. Assuming that utilization z is stable over time we can simplify the formula as:

$$opex_{zSeries}(h_\tau, z) = k_e \times h_\tau \times z \tag{1}$$

where k_e is the cost of electricity per hour of operation and h_τ the total hours of operation in τ. For the latter, as discussed in [2], the cost calculation function can be inferred by the (publicly available) pricing policies in each cloud provider's Web site. In particular for the 'EC2 m1.large' Reserved Instances offering in the European Region using Linux, operational expenses (in US dollars) can be written analytically as $opex_{EC2\_m1.large}(h_\tau) = k_{initial} + h_\tau \times k_{perHour}$, or in prices of December 2013 [7]:

$$opex_{EC2\_m1.large}(h_\tau) = \begin{cases} 243 + h_\tau \times 0.17 & \text{with 1-year contract} \\ 384 + h_\tau \times 0.134 & \text{with 3-year contract} \end{cases} \tag{2}$$

where h_τ, as in the case of $opex_{zSeries}$, is the total amount of hours that the offering has been used in period τ, and $k_{initial}, k_{perHour}$ the initial cost for the reservation of instances and the cost of the offering per hour of use, respectively.

[7] As defined in `http://aws.amazon.com/ec2/pricing/#reserved`

Assuming that the application owner has a fixed budget k_{max} that is allowed to spend on operational expenses, it follows from Equations 1 and 2 that

$$u(T^{(1)}) = u^{(1)}(k_{max}, h_\tau, z) = k_{max} - (opex_{zSeries}(h_\tau, z) + opex_{EC2\_m1.large}(h_\tau))$$

and $A^{(1)} = \{k_{max}, h_\tau, z\}$.

The utility function $u^{(i)}$ and arguments set $A^{(i)}$ for each topology in $\mathcal{V}$ can be defined in a similar fashion. In principle, if utility (and therefore optimization of the distribution) considers more than one dimensions, e.g. cost and deployment time, then it can be written as a weighted and normalized sum of functions. However for the purposes of this work, we restrict the discussion to single-dimension optimization and leave the multiple dimension problem for future work.

4.2 Optimal Topology Selection

Following from the above, optimizing the distribution of an application for a given set of parameters (values) $\mathcal{P}$, which constitutes the *application profile*, can therefore be viewed as the (partial) ordering of the set $\mathcal{V}$ of the viable topologies for the application based on the evaluation of their utility functions for (potentially a subset of) $\mathcal{P}$.

More specifically, starting from the partial ordering

$$eval(u(T^{(1)}), \mathcal{P}) \geqslant eval(u(T^{(2)}), \mathcal{P}) \geqslant \ldots \geqslant eval(u(T^{(m)}), \mathcal{P}), m = |\mathcal{V}|$$

and given the assumption that the utility functions $u^{(i)}$ are monotonic in nature, then ordering the evaluation of these functions allows for the ordering of the viable topologies, e.g. and without loss of generality:

$$eval(u(T^{(1)}), \mathcal{P}) \leq \ldots \leq eval(u(T^{(m)}), \mathcal{P}) \Rightarrow T^{(1)} \leq \ldots \leq T^{(m)}. \quad (3)$$

Given therefore an application profile $\mathcal{P}$ and by identifying the appropriate utility functions in the set $\mathcal{U} \subset \mathbb{F}$ for the set of viable topologies $\mathcal{V}$, the optimal distribution of the application can be selected by Equation 3. $\mathcal{V}$ can be first trimmed down to a smaller size (which would allow for better performance given the number of evaluations and comparisons required) by applying a set of *filter* functions $\sigma^{(1)} \circ \cdots \circ \sigma^{(k)}(T^{(i)})$ where

$$\sigma^{(j)}(T^{(i)}) = \begin{cases} T^{(i)} & \text{if condition}(j)\text{=true,} \\ \emptyset & \text{otherwise.} \end{cases}$$

with condition(j),$1 \leq j \leq k$ written as logical formulas e.g. $Apache\_Tomcat \in T^{(i)}$ to denote whether an Apache Tomcat typed node appears in $T^{(i)}$.

In terms of automating this process, optimally distributing an application across cloud offerings with respect to one or more dimensions (e.g. operational expenses) and for a set of predefined constraints $\mathcal{C}$ (e.g. the data layer of the

application must remain on premises) can be decomposed into the following steps:

1. If no μ-topology is available, then construct and merge the α-topology of the application with an available γ-topology.
2. Generate the set of viable topologies $\mathcal{V}$ from the μ-topology by traversing the typed graph with inheritance.
3. Prune down $\mathcal{V}$ by iteratively applying the filter functions $sigma^{(j)}$ for each $j \in \mathcal{C}$.
4. For each viable topology $T^{(i)}$ remaining in $\mathcal{V}$, identify the utility function $u^{(i)}$ that is relevant for the optimization dimension.
5. Construct the set of parameter values $\mathcal{P} = \bigcup_i^{|\mathcal{V}|} P^{(i)},\ \forall u^{(i)} \in \mathcal{U}$.
6. Calculate $r^{(i)} = eval(u(T^{(i)}), \mathcal{P}),\ \forall T^{(i)} \in \mathcal{V}$ (or equivalently $\forall u^{(i)} \in \mathcal{U}$).
7. Select the topology corresponding to the value $max\{r^{(i)} \mid 1 \leq i \leq |\mathcal{V}|\}$.

In the following section we demonstrate how these steps can be applied in practice for the Web Shop application presented in Section 2 in order to optimize the distribution of the application in terms of operational expenses and with a set of architectural constraints.

5 Evaluation

The evaluation of our methodology presented in this work is based on the Web Shop application described in Section 2. More specifically, as discussed in the previous sections, Fig. 2 outlines some of the alternative viable topologies for the application, such as hosting the database on AWS RDS instead of a virtual machine, and so on. In the following we show how the proposed method can be used to optimize the distribution of the application w.r.t. operational expenses. For evaluation purposes we came up with a synthetic application profile $\mathcal{P}$ for the Web Shop application to be used in the fifth step of the optimization process discussed in the previous section. This profile, shown in Table 1, ensures that alternative viable topologies hosted on different Cloud providers are comparable. Furthermore, for reasons of completeness we also considered similar offerings to

Table 1. Web Shop Application Profile Parameters

Parameter		**Value**
h_τ	Hours of Usage	5,040
τ	Months of Usage	15
$n_{I/O}$	Number of I/O Ops	5,000
$d_{storage}$	Storage in GB	5,000
d_{egress}	Outgoing Traffic in GB	50,000
loc	Infrastructure Location	Europe

AWS EC2 and RDS provided by Windows Azure and Rackspace that we do not show explicitly in Fig. 2 but nevertheless include in the following discussion.

In terms of the proposed method, Fig. 2 already illustrates a μ-topology for the application containing offerings from only one cloud provider (AWS). Traversing the graph in the figure and generating viable topologies follows from the use of inheritance clan relation in the choice of e.g. Linux instead of Windows for the back end of the application. In order to limit the search space we define the constraint set $\mathcal{C} = \{c_{front}, c_{provider}\}$ where $c_{front} = IBM\_zSeries \in T^{(i)} \wedge \exists \text{path} \in T^{(i)} : WebShop\_FrontEnd \xrightarrow{*} IBM\_zSeries$, i.e. the application front end must be deployed on an IBM zSeries server, and $c_{provider} = (AWS\_EC2 \in T^{(i)} \vee AWS\_RDS \in T^{(i)}) \oplus (Azure\_VM \in T^{(i)} \vee Azure\_SQL \in T^{(i)}) \oplus \ldots$, i.e. cloud offerings only from one provider at a time are allowed to avoid latency between providers. The remaining viable topologies in $\mathcal{V}$ after applying the filter functions $\sigma^{front} \circ \sigma^{provider}$ for the offerings of the three cloud providers (AWS, Azure and Rackspace) are depicted in Table 2. For example, $T^{(1)}$ is the initial topology, $T^{(2)}$ is its variation that uses two smaller Amazon EC2 m1.medium instances, both with a different operating system, for a separate deployment of the Web Shop's back end and persistence tiers, etc.

The equivalent provider offerings that can be used, and the monetary cost projected for each $T^{(i)}$ for the application profile $\mathcal{P}$ is calculated by utilizing the Nefolog system [24]. For a given application usage profile and time interval, Nefolog provides cloud offerings matching and cost estimation capabilities as RESTful services. For the given application profile and viable topologies, we first used the *Offerings Matcher* service to identify a set of offerings from the two additional to AWS providers (Windows Azure and Rackspace) that are equivalent to the AWS offerings used in Fig. 1. We then invoked the *Cost Calculator* service for each of the identified offerings, and the results of the service invocation was stored in an $N \times M$ matrix, where N is the Cost Calculator operation query parameters (using the application profile in Table 1) and M the projected cost of each Cloud offering for each $T^{(i)}$ topology. The total cost of operational expenses $opex^{(i)}$ for each $T^{(i)}$ with application profile parameters from Table 1 is shown in Table 2.

By denoting with $opex_{max}$ the maximum calculated cost ($154,134 for a Windows server and database solution in Rackspace for 15 months of use), choosing for utility function

$$u^{(i)}(h_\tau, \tau, n_{I/O}, d_{storage}, d_{egress}, loc) = opex_{max} - opex^{(i)}(h_\tau, \tau, \ldots)$$

(so that the utility function is monotonically decreasing with the cost) and by using the application profile in Table 1 it can be seen that:

$$r^{(9)} < r^{(8)} < r^{(7)} < r^{(6)} < r^{(3)} < r^{(5)} < r^{(1)} < r^{(2)} < r^{(4)} \Rightarrow$$
$$T^{(9)} < T^{(8)} < T^{(7)} < T^{(6)} < T^{(3)} < T^{(5)} < T^{(1)} < T^{(2)} < T^{(4)}$$

From this it can therefore be concluded that for the given application profile, the *optimal w.r.t. operational expenses distribution of the Web Shop application*

Table 2. Cost Analysis for a Subset of the Viable Topologies of the Web Shop Application

Topology	Provider	Cloud Offerings Used	Total Cost
$T^{(1)}$	AWS	Backend: *EC2 m1.large* (Windows) ProductDB: *in the same VM*	$77,037
$T^{(2)}$	AWS	Backend: *EC2 m1.medium* (Windows) ProductDB: *EC2 m1.medium* (Linux)	$75,942
$T^{(3)}$	AWS	Backend: *EC2 m1.medium* (Windows) ProductDB: *RDS db.m1.medium* (MySQL)	$84,904
$T^{(4)}$	Azure	Backend: *VM Large A3* (Windows) ProductDB: *in the same VM*	$75,504
$T^{(5)}$	Azure	Backend: *VM Medium A2* (Windows) ProductDB: *VM Medium A2* (Linux)	$84,504
$T^{(6)}$	Azure	Backend: *VM Medium A2* (Windows) ProductDB: *SQL ProductDB* (Microsoft SQL)	$86,139
$T^{(7)}$	Rackspace	Backend: *Cloud Server 8GB* (Windows) ProductDB: *in the same VM*	$96,351
$T^{(8)}$	Rackspace	Backend:*Cloud Server 4GB* (Linux) ProductDB: *Cloud Server 4GB* (Linux)	$105,432
$T^{(9)}$	Rackspace	Backend: *Cloud Server 4GB* (Windows) ProductDB: *Cloud Database 4GB* (MySQL)	$154,134

is represented by the viable topology $T^{(4)}$, i.e. using a single Windows Azure VM to host both the back end and persistence tier of the application. Looking only at topologies that are deployed on AWS offerings, it can also be seen that it is cheaper to distribute the application back end and persistence tier across two smaller VMs instead of a larger one. It is obvious from the above that elasticity is not considered in these results. Adding to the application profile the use of multiple VMs to cope with variation in demand would potentially result in a different ordering of the topologies. However, adding this capability to the presented framework is at this point in time future work.

6 Related Work

As discussed in the introductory section, decision support-oriented approaches like Kingfisher [21], CloudGenius [16], CloudAdoption [10] and MDSS [2] focus on assisting application designers in migrating their applications to the cloud. The main focus of these works is on optimal cloud offering selection for a given application that is essentially treated as a monolithic artifact. These approaches are therefore limited in their usefulness in light of multiple potential application topologies considering the distribution of the application across offerings.

The Cloud Blueprinting approach [19,20] defines a blueprint as an abstract description of cloud service offerings that facilitates the selection, customization and composition of cloud services into service-based applications. Blueprint templates allow the application developers to define the requirements of the application in terms of functional capabilities, QoS characteristics, and deployment and provisioning resources as target blueprints. In this respect, target blueprints are equivalent to α-topologies. However, the blueprinting approach is geared towards matching requirements with available solutions in a repository, having no equivalent concept to γ-topology, and lacking therefore the ability to generate viable topologies for an application.

The proposed approach shares a similarity with other existing works, in the sense that they use application topologies to optimize for a set of dimensions usually involving operational expenses. For example, the work in [18] presents DADL, a language to describe the architecture, behavior and needs of a distributed application to be deployed on the cloud, as well as describing available cloud offerings for matching purposes. Similarly, in [3], the authors propose an approach that matches and dynamically adapts the allocation of infrastructure resources to an application topology in order to ensure SLAs. CloudMig [8] is another approach that builds on an initial topology of the application that is adapted through model transformation in order to optimize the distribution of the application across cloud offerings. The optimization in this case also focuses on SLA compliance in a trade-off relation to operational expenses. The approach in [17] uses a Palladio-based application topology model in order to distribute an application across different cloud providers aiming at optimizing for availability and operational expenses.

Nevertheless, all of the above approaches assume that the application topology is already known (and fixed), and are restricted to VM-based IaaS solutions. Our approach takes into account also non-VM cloud offerings like DBaaS offerings, and allows for the dynamic generation of acceptable application topologies, which may also include alternative application stacks based on the richness of the γ-topology of the application. This observation also applies to the most relevant for our proposal work, the MOCCA framework [13] which also discusses the optimization of the application to cloud offerings based on introducing variability points in the application topology.

In terms of non-cloud scenarios, optimization of the application distribution has been discussed as part of various approaches like [11,14] and [15]. These approaches focus on performance engineering that is not discussed in the scope of this work, and in this sense they can be useful in extending the current work. However, it is necessary to evaluate first to which extend their underlying application topology models can be leveraged for cloud applications.

7 Conclusions

The previous sections motivated the need for moving away from the practice of thinking of applications as monolithic stacks to be deployed in one VM, either in the cloud or in in-house servers. Multi-tiered applications in particular can be

seen as an aggregation of application-specific components, middleware solutions supporting these components, and of the underlying infrastructure in which they are being deployed and provisioned. The availability of cloud offerings beyond the VM-oriented IaaS solutions, and the proliferation of cloud services competing for market share create both opportunities and challenges for application engineers. Optimizing the distribution of an application across potentially multiple cloud offerings is therefore an important requirement for reaping the benefits of the cloud computing paradigm.

Toward this goal, in this work we proposed a theoretical framework that supports decision making in identifying the optimal application distribution. For this purpose we approached application topology models as typed graphs and leveraged existing work on introducing inheritance relationships between nodes in them. Using the concept of type graph with inheritance we introduced the concept of μ-topologies as an abstraction over two distincts parts of an application topology: α-topology, the application specific aspect of the model, and γ-topology, being reusable across different applications. We then showed how μ-topologies can be used to generate viable topologies as alternative deployment scenarios for the application, taking into account different types of cloud offerings. Based on this foundation we then proposed a generic definition of the optimization distribution problem as a mapping from the application topology domain to the set of real numbers, which allows for the (partial) ordering of alternative solutions and therefore simplifies the optimization.

One clear deficiency of the proposed approach is the ability to scale with the number of nodes in the μ-topology. Rich γ-topologies are of course necessary in order to be able to generate as many alternative solutions as possible, but the richer the γ-topology used is, the larger the space that needs to be searched through. Realizing the topology generation and selection algorithm discussed in Section 4.2, evaluating its complexity for real-world examples and dealing with potential scalability problems is in our immediate future plans.

For purposes of illustrative examples and evaluation throughout this work the focus was on operational expenses, and showing how the most cost-efficient application topology can be identified. Beyond offering the opportunity for concrete examples and easily verifiable results with publicly available information, this choice was also based on what is perceived to have driven the growth of cloud computing, i.e. cost reduction due to economies of scale on the provider side. However, the presented framework is not limited to optimization for operational expenses. As discussed in the previous section, there are a number of available performance engineering approaches that can be used to extend this work, e.g. [11,14,15]. To this goal, investigating the role of the workload of the application as reflected in its distribution across (topological) nodes is of interest for future work, as well as the effect of network latency between cloud offerings of the same and different providers (see also the discussion in [1] building on [9]). Finally, evaluating the impact of different scalability strategies, expanding on works like [21,22], is as mentioned in Section 5 an example of an additional optimization dimension to be introduced in future work.

Acknowledgment. This work is partially funded by the FP7 EU-FET project 600792 ALLOW Ensembles.

References

1. Andrikopoulos, V., Binz, T., Leymann, F., Strauch, S.: How to Adapt Applications for the Cloud Environment. Computing 95(6), 493–535 (2013)
2. Andrikopoulos, V., Song, Z., Leymann, F.: Supporting the migration of applications to the cloud through a decision support system. In: Proceedings of the 6th IEEE International Conference on Cloud Computing (CLOUD 2013), pp. 565–572. IEEE Computer Society (2013)
3. Antonescu, A.F., Robinson, P., Braun, T.: Dynamic topology orchestration for distributed cloud-based applications. In: Second Symposium on Network Cloud Computing and Applications (NCCA), pp. 116–123 (2012)
4. Armbrust, M., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)
5. Bardohl, R., Ehrig, H., De Lara, J., Runge, O., Taentzer, G., Weinhold, I.: Node type inheritance concept for typed graph transformation. Tech. Rep. 2003-19, TU Berlin (2003), `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.9257&rep=rep1&type=pdf`
6. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services, pp. 527–549. Springer (2014)
7. Brandtzæg, E., Mohagheghi, P., Mosser, S.: Towards a domain-specific language to deploy applications in the clouds. In: The Third International Conference on Cloud Computing, GRIDs, and Virtualization, Cloud Computing 2012, pp. 213–218. IARIA (2012)
8. Frey, S., Hasselbring, W.: The cloudmig approach: Model-based migration of software systems to cloud-optimized applications. International Journal on Advances in Software 4(3 and 4), 342–353 (2011)
9. Gray, J.: Distributed Computing Economics. Queue 6(3), 63–68 (2008)
10. Khajeh-Hosseini, A., Greenwood, D., Smith, J.W., Sommerville, I.: The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise. Software: Practice and Experience 42(4), 447–465 (2012)
11. Koziolek, A., Koziolek, H., Reussner, R.: Peropteryx: automated application of tactics in multi-objective software architecture optimization. In: Proceedings of the joint ACM SIGSOFT QoSA and ISARCS, pp. 33–42. ACM (2011)
12. de Lara, J., Bardohl, R., Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Attributed graph transformation with node type inheritance. Theoretical Computer Science 376(3), 139–163 (2007)
13. Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving applications to the cloud: An approach based on application model enrichment. International Journal of Cooperative Information Systems 20(03), 307–356 (2011)
14. Malek, S., Medvidovic, N., Mikic-Rakic, M.: An extensible framework for improving a distributed software system's deployment architecture. IEEE Transactions on Software Engineering 38(1), 73–100 (2012)

15. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering, pp. 105–116. ACM (2010)
16. Menzel, M., Ranjan, R.: Cloudgenius: decision support for web server cloud migration. In: Proceedings of the 21st International Conference on World Wide Web, WWW 2012, pp. 979–988. ACM, New York (2012)
17. Miglierina, M., Gibilisco, G., Ardagna, D., Di Nitto, E.: Model based control for multi-cloud applications. In: 5th International Workshop on Modeling in Software Engineering (MiSE), pp. 37–43 (2013)
18. Mirkovic, J., Faber, T., Hsieh, P., Malaiyandisamy, G., Malaviya, R.: DADL: Distributed Application Description Language. Tech. Rep. ISI-TR-664, USC/ISI (2010), ftp://www.isi.edu/isi-pubs/tr-664.pdf
19. Nguyen, D.K., Lelli, F., Papazoglou, M.P., Van Den Heuvel, W.J.: Blueprinting approach in support of cloud computing. Future Internet 4(1), 322–346 (2012)
20. Papazoglou, M.P., van den Heuvel, W.: Blueprinting the cloud. Internet Computing 15(6), 74–79 (2011)
21. Sharma, U., Shenoy, P., Sahu, S., Shaikh, A.: Kingfisher: Cost-aware elasticity in the cloud. In: Proceedings of INFOCOM 2011, pp. 206–210. IEEE (2011)
22. Suleiman, B., Sakr, S., Jeffery, R., Liu, A.: On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. Journal of Internet Services and Applications, 1–21 (2011)
23. Walker, E.: The real cost of a cpu hour. Computer 42(4), 35–41 (2009)
24. Xiu, M., Andrikopoulos, V.: The Nefolog & MiDSuS Systems for Cloud Migration Support. Technical Report 2013/08, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany (November 2013), http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2013