Utility-based Decision Making for Migrating Cloud-based Applications

SANTIAGO GÓMEZ SÁEZ, University of Stuttgart
VASILIOS ANDRIKOPOULOS, University of Groningen
MARINA BITSAKI, University of Crete
FRANK LEYMANN, University of Stuttgart
ANDRÉ VAN HOORN, University of Stuttgart

Nowadays, cloud providers offer a broad catalog of services for migrating and distributing applications in the cloud. However, the existence of a wide spectrum of cloud services has become a challenge for deciding where to host applications, as these vary in performance and cost. This work addresses such a challenge, and provides a utility-based decision support model and method that evaluates and ranks during design time potential application distributions spanned among heterogeneous cloud services. The utility model is evaluated using the MediaWiki (Wikipedia) application, and shows an improved efficiency for selecting cloud services in comparison to other decision making approaches.

CCS Concepts: •**Computer systems organization** → **Cloud computing;** •**Software and its engineering** → **Designing software; Software design engineering;**

Additional Key Words and Phrases: Cloud Application Topologies, Cloud Services Selection, Utility Theory, Decision Making

ACM Reference Format:
Santiago Gómez Sáez, Vasilios Andrikopoulos, Marina Bitsaki, Frank Leymann, and André van Hoorn, 2016. Utility-based Decision Making for Migrating Cloud-based Applications. *ACM Trans. Internet Technol.* 1, 1, Article 1 (January 2016), 21 pages.
DOI: 0000001.0000001

1. INTRODUCTION

The rapid growth of cloud providers and solutions has opened in the last years a broad umbrella of possibilities for partially or completely migrating applications to *Everything-as-a-Service* (*aaS) offerings [Andrikopoulos et al. 2013]. Besides, the materialization and usage of DevOps principles in both research and industry domains seem to be more and more evident, as these allow to rapidly develop, provision, deploy, and adapt applications in the cloud [Humble and Molesky 2011; Bass et al. 2015].

Focusing in this work on business applications and the cloud model defined in the NIST definition of cloud computing [Mell and Grance 2011], cloud computing is basically a key enabler of rapid business growth and transformation, as it allows to reduce costs while ensuring rapid deployment and scalability properties [Council 2013]. However, the heterogeneity of cloud services and providers has progressively become a challenge for migrating applications to the cloud, as (i) the performance of cloud services typically fluctuates and varies w.r.t. the type of cloud service and provider [Gómez Sáez et al. 2015],

This research is founded by the German DFG projects SitOPT (610872) and Declare (HO 5721/1-1).
Author's addresses: Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Germany; Marina Bitsaki, Department of Computer Science, University of Crete, Greece; André van Hoorn, Reliable Software Systems Research Group, Institute of Software Technology, University of Stuttgart, Germany.

DOI: 0000001.0000001

and (ii) the fluctuation of the application workload has an impact on the application's overall performance [Gómez Sáez et al. 2014]. In particular, applications cannot be re-engineered for the cloud using traditional methods and techniques, due to the fundamental properties of cloud infrastructures, such as multi-tenancy and elasticity [Harms and Yamartino 2010]. Informally defined, cloud-based applications *rely on one or more cloud services in order to be able to deliver their functionality to their users*. Application engineers must therefore rethink how to (re)design and provision their applications to be enabled for the cloud, as its simple packaging in virtual machines does not exploit the full spectrum of cloud services. Moreover, design tools and decision making techniques were not originally designed for the cloud, including decision support mechanisms and tools to assist in the optimal selection and configuration of cloud services [Jamshidi et al. 2013].

The *SCAR Framework (SCARF)* is the main pillar in our research agenda [Gómez Sáez et al. 2014; Gómez Sáez et al. 2016], and is geared towards *assisting application developers to efficiently (re)distribute their application components spanned among multiple cloud offerings to cope with business objectives and variable performance demands*. This work focuses on migrating applications that follow the layered architectural pattern [Fowler 2002], such as the three-layered Web shop application evaluated in [Andrikopoulos et al. 2014]. In particular, this work materializes the decision making mechanism in SCARF, by using utility theory for the cost- and performance- efficient distribution of cloud-based applications.

Utility, defined as the perceived satisfaction when consuming a good or service, emerged in the economics domain towards understanding decisions and assisting decision making processes over goods and services [Marshall 2009]. More specifically, utility is a measure of preferences over a set of good of services [Marshall 2009], which is typically used in game theory and decision making mechanisms, e.g., in multi-attribute utility theory [Keeney and Raiffa 1993]. Utility has also been utilized for optimizing the allocation of computational [Minarolli and Freisleben 2011a] and storage resources [Strunk et al. 2008], and in this work has one major goal: *assisting business and IT experts in the decision making tasks when spanning their applications among multiple and heterogeneous cloud services and providers*. Utility functions can assist in evaluating the trade-off between application requirements, such as the ones related to the cost and performance. The contributions of this paper are:

(1) An extension of the SCARF life cycle introduced in our previous work [Gómez Sáez et al. 2016], which focuses on the performance- and cost-efficient distribution of applications, by means of incorporating the decision making phases and using utility theory as the basis,
(2) a formal utility model serving as the underlying decision making mechanism to assist in the distribution of cloud-based applications consuming different cloud services, and
(3) a first evaluation of such a model using a two-layered MediaWiki[1] application, the Wikipedia realistic workload and data, its financial reports, and under different single-provider distribution scenarios.

Evaluation results show a better accuracy when using the proposed utility model for the decision making tasks to distribute applications among cloud services. The remainder of this paper is structured as follows. Sec. 2 summarizes relevant concepts this work builds upon. The utility model for optimizing the distribution of cloud-based applications is presented in Sec. 3.1, which are subsequently evaluated in Sec. 4. Sec. 5 presents the limitations of our approach, Sec. 6 introduces related works, and Sec. 7 concludes with future research works.

2. (RE)DISTRIBUTION OF CLOUD-BASED APPLICATIONS

This work builds upon two pillars: (i) the design and distribution of cloud-based applications, and the (ii) economic models used for decision making of cloud-based applications. In this section, we tackle the former by means of introducing a (i) cloud topology model used

[1]MediaWiki: https://www.mediawiki.org

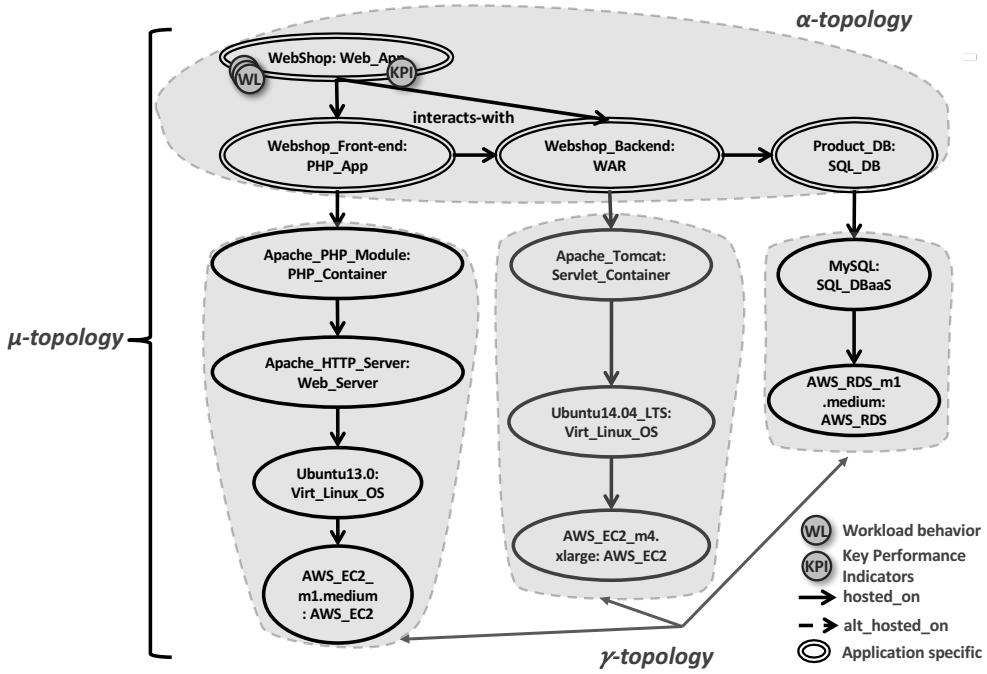


Fig. 1: Topology Model - WebShop Example

as the underpinning support for the modeling of reusable application topologies, which is used in (ii) SCARF, consisting of a life cycle and the Systematic Cloud-based Application (Re)Distribution Method (SCARM) for the (re)distribution of applications in the cloud.

2.1. Cloud-based Application Topology Model

A cloud-based application topology model is a labeled acyclic graph depicting the application stack, by means of representing application components and services as nodes, and the relationships among them as a set of edges [Andrikopoulos et al. 2014]. Fig. 1 depicts a three-tiered Web shop application topology, comprising a PHP-based front-end, a Java-based back-end, and a SQL database back-end. Towards empowering the reusability of topology models among applications, we proposed to model an application topology as a typed topology graph model, which can be partitioned into a graph model that depicts the application-specific (α-topology) and non-application specific γ-topology [Andrikopoulos et al. 2014]. α-topologies represent the components that are unique and specific for each application, e.g., the front- and back-ends of the Web shop in Fig. 1, while the γ sub-topologies depict application non-specific components, such as middleware components like an Apache Web server or a MySQL DBaaS offering. α-topologies are the result of alternative architectural decisions, which result into different topology models. γ sub-topologies are intended to be used in further application topologies, e.g., using the Apache Web server γ-topology of Fig. 1 in another PHP-based application. The utilization of inference in the typed topology graph model allows to discover one or multiple viable distributions of the application, denoted as μ-topologies in Fig. 1.

So far, we introduced a topology model comprising the functional aspects of the application. Incorporating in the topology non-functional aspects, such as performance and cost constraints, can serve as a filtering mechanism for discovered μ-topologies, e.g., in order to

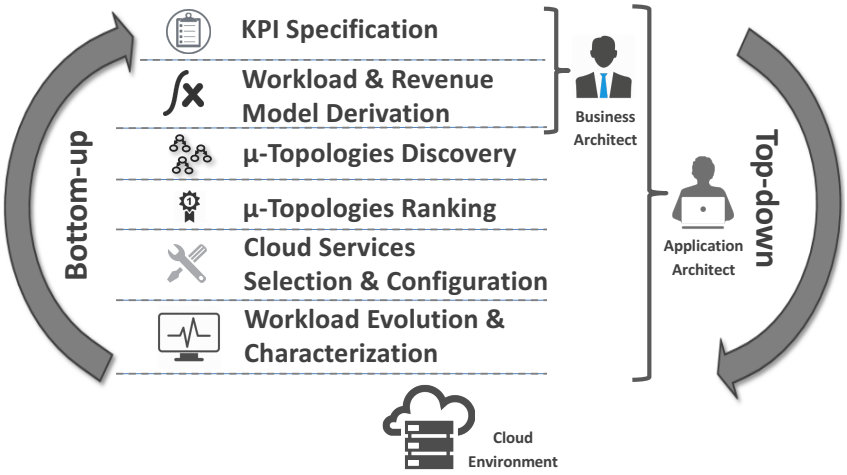


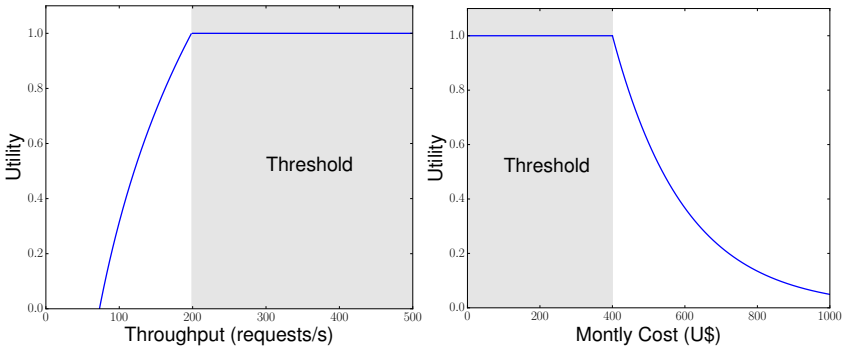
Fig. 2: Optimal Distribution of Cloud-based Applications - Life-Cycle

trim down the number of viable application distributions [Andrikopoulos et al. 2014]. For this, a cost- and performance-aware topology model incorporating the notions of cost and performance was introduced in [Gómez Sáez et al. 2016]. The introduced topology model is fundamental for the utility-based analysis carried on in the remainder of this paper, as it is used as the basis for the decision making tasks related to optimally selecting a specific distribution of an application (μ-topology) [Gómez Sáez et al. 2014; Gómez Sáez et al. 2016].

2.2. SCARF: The SCAR Framework

The distribution of applications in cloud environments is typically a non-trivial task, due to the diversity of cloud offerings, providers, and the heterogeneous characteristics among them. Architecting cloud-based applications in a cost- and performance-efficient manner requires to consider (i) the difference between the required and offered performance, and (ii) the application workload behavior evolution.

The first ingredient in SCARF consists of (i) a life-cycle (extended from [Gómez Sáez et al. 2016]) depicted in Fig. 2, and (ii) the Systematic Cloud-based Application (Re)Distribution Method (SCARM) introduced in [Gómez Sáez et al. 2016]. Since migration of applications to the cloud often involves the interaction of business leaders and IT professionals [Laverty et al. 2014], we consider two main actors in the design and development of distributed cloud-based applications: the (i) *Application Architect*, responsible for the architectural design and planning of the application distribution, and the application profile, and the (ii) *Business Architect*, responsible for the analysis and creation of business KPIs and plans, such as the analysis and derivation of revenue models and objectives for the different business lines. Focusing on the life-cycle phases, the first phase consists of the *KPI Specification*, by means of defining and specifying the business and operational requirements. For the MediaWiki application depicted in Fig. 1, a KPI business requirement can be related to increasing the monetary incomes by 15%, while an operational requirement can be related to reducing the maintenance costs by 40%. In the *Workload & Revenue Model Derivation* phase, application and business architects define and derive the workload behavioral and revenue models, respectively. Workload models are defined as a *probabilistic model representing different potential workload behaviors, each depicting the arrival rate of users and transactions that impact the application state* [Gómez Sáez et al. 2016]. In the *μ-topologies Discovery* phase, a set of viable application distributions is constructed. The definition of the application profile



(a) QoS-aware Utility for the Application's Throughput

(b) Cost-aware Utility for the Application's Operational Costs

Fig. 3: Utility Calculation & Trend Example

serves as the basis in the *μ-topologies Ranking* phase, which ranks each viable topology in the μ-topology based on the expected utility, and using utility functions as the basis. Fig. 3 depicts how the shape of a utility function could look like when considering the QoS (Quality of Service) in terms of the application's throughput, and cost in USD Dollars.

Focusing on the QoS, there may exist an increase of the utility as the throughput increases until a certain requirement threshold point, as the application is capable of serving more requests (see Fig. 3). However, from such a point, the application's utility may not significantly vary, as the requirement is completely fulfilled, and therefore the utility maintains constant. Focusing on the cost, it actually seems to behave in an adverse manner, as going from a IaaS *General Purpose* VM to a *Compute* or *Memory Optimized* may negatively impact the utility, since the operational costs increase. For the Web shop application depicted in Fig. 1, provisioning a *high I/O* VM instance may negatively impact its utility, due to the compute and memory intensive nature of such application. However, provisioning *compute and memory intensive* VM instances during peak periods may significantly benefit the overall revenues.

The utility-based evaluation of μ-topology models in SCARF consists of analyzing the trade-off between the performance and the cost for viable application distributions. The utility model assists in the decision making tasks in the *Selection & Configuration* phase. During the production phase of the application, the *Workload Evolution & Characterization* phase consists of retrieving performance data to analyze and build the workload and performance evolution knowledge, using, e.g., monitoring techniques. Such knowledge can be leveraged, in conjunction with the business revenue models, in the utility analysis, to move towards an optimal distribution of cloud-based applications.

SCARM constitutes the second pillar in SCARF, and supports the life-cycle depicted in Fig. 2 [Gómez Sáez et al. 2014]. Fig. 4 depicts the seven tasks of SCARM, and characterizes them as manual (driven by application architects), automatic (performed by the tooling support), and synergistic (as the interaction of both). Application architects are responsible in SCARM for manually (i) modeling the application topology (*Modeling*), which is (ii) enriched with business and operational requirements, and workload characteristics (*Enrichment*). The enriched application topology model is then automatically processed in the (iii) *KPIs & WL Analysis* task. In particular, application KPIs and workload attributes are analyzed in order to (iv) discover compatible γ-topologies, and to construct and evaluate alternative viable distributions of the application components, i.e. depicted through their μ-topologies (*Discovery & Evaluation*). The evaluation of the constructed viable distributions is performed

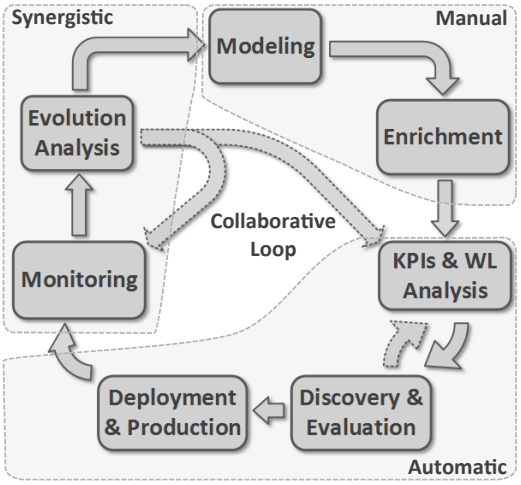


Fig. 4: SCARM: Systematic Cloud-based Application (Re)Distribution Method

by calculating the utility of each μ-topology. Utility is leveraged towards ranking the different μ-topologies to assist application architects when selecting a viable application distribution.

The (v) *Deployment & Production* task consist of provisioning the application stack using existing orchestration technologies, such as the OpenTOSCA container [Binz et al. 2013]. During the production phase of the application, the following synergistic decision making tasks take place: the (vi) *Monitoring* task captures real performance metrics, which are leveraged by application architects to (vii) analyze the evolution of the application performance and workload behavior in the *Evolution Analysis* task. The *Collaborative Loop* in SCARM enables application architects to decide among redistributing the application components or to continuously capture performance knowledge that can be exploited in subsequent iterations. This work focus on the utility-based evaluation step of SCARM, i.e., (iv), by means of using utility theory as the underlying model for the decision making mechanism for distributing cloud-based applications.

3. UTILITY MODEL IN SCARF

This section presents a formal utility model geared towards the profitable distribution of cloud-based applications in SCARF, using the Web shop application previously introduced as example and depicting further distribution alternatives (see Fig. 1). For the scope of this work, the SCARF utility model *quantitatively represents the monetary cost and performance trade-off of a viable distribution of an application (depicted as a μ-topology) spanning different cloud services for a time period.* This section first defines the variables in the model, which are then used as the basis to formalize the utility functions.

3.1. Preliminary Definitions

Let's first define the life time of an application as the family of sets $\Psi = \{\Psi_1, ..., \Psi_m\}$, where each Ψ_j, $j = 1, ..., m$ is an ordered discrete time interval $\{t_1, ..., t_n\}$ that corresponds to a viable topology (μ-topology) used to provision and deploy an application. For example, if an application is redistributed on a weekly basis, then each Ψ_m would be comprised by the initial and end date of a concrete distribution, e.g., $\Psi_1 = \{Mo\,18.7.2016, ..., Su\,24.7.2016\}$, Since a cloud-based application viable μ-topology is decomposed into an application specific α-topology and multiple non-application specific (and reusable) γ-topology models (see Sec. 2.1), let's define:

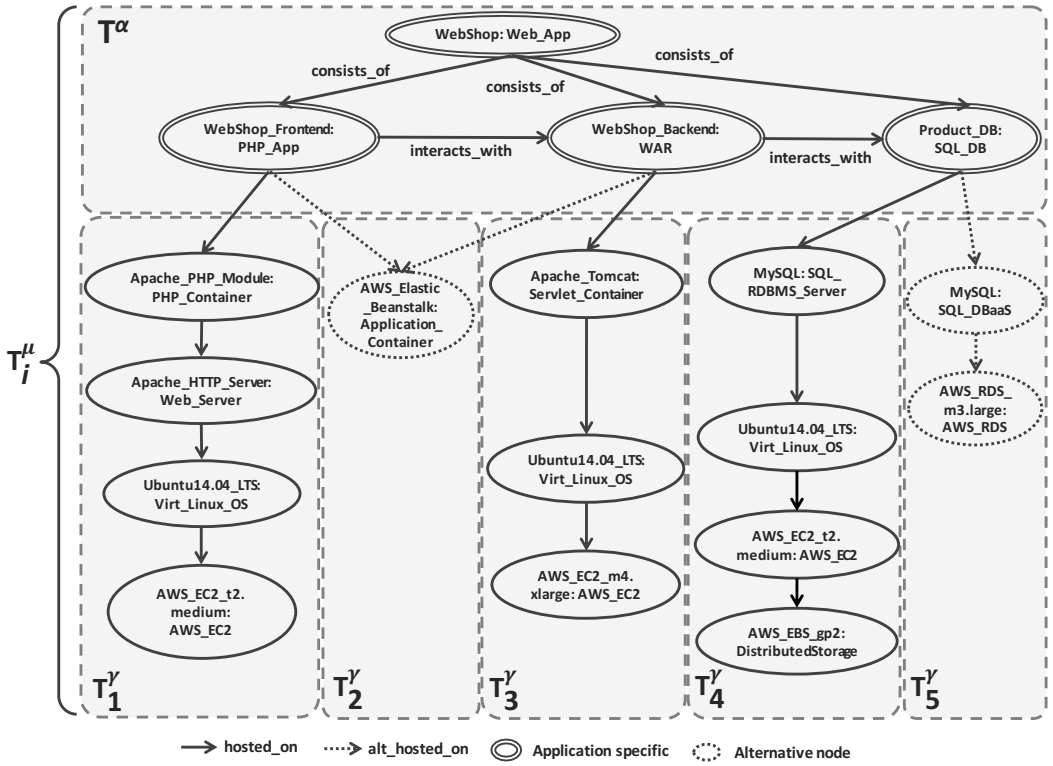


Fig. 5: T^{α}, T^{γ}, and T^{μ} topologies for a sample Web Shop Application

— T^{α} as the set of application specific α-topology models $\{T_1^{\alpha}, ..., T_p^{\alpha}\}$. T^{α} contains all α-topology models, that are a result of architectural decisions made during the lifetime of applications.
— T^{γ} as the set of all available and reusable application non-specific γ-topologies $\{T_1^{\gamma}...T_q^{\gamma}\}$, where each T^{γ} represents the underlying resources, such as middleware or cloud services, that can be used to host one or multiple application components (in other words, T^{γ} can be interpreted as a repository of all possible reusable γ-topologies).
— The set of viable application μ-topologies as $T^{\mu} = \{T_1^{\mu}, ..., T_i^{\mu} \mid |T^{\mu}| = |\Psi| \ \wedge \ \forall \, T_j^{\mu}, j = 1, ..., i \ \exists \, t_{map} : T_j^{\mu} \ \mapsto \ \Psi_m \ \wedge \ f_{dist} : \ T^{\alpha} \times T^{\gamma} \ \rightarrow \ T_i^{\mu}\}$, where t_{map} is a function mapping each T_j^{μ} with a concrete time interval Ψ_m in the application's life time, and f_{dist} is a constructor function, which builds a concrete T_j^{μ} given its application specific α-topology T_p^{α} and the set of available reusable γ-topologies T^{γ}. f_{dist} relies on the usage of graph morphisms presented in [Andrikopoulos et al. 2014].
— The subset $\tau^{\gamma} \subseteq T^{\gamma}$ contains multiple sets τ_v^{γ}, $v = 1, ..., q$, where each set encloses the $T_u^{\gamma}, u = 1, ..., q$ topologies that are part of a viable topology of an application $T_j^{\mu} \, \epsilon \, T^{\mu}, j = 1, ..., i$.

Taking the Web shop application μ-topology depicted in Fig. 5 for a time interval Ψ_1, the set T^{α} contains only one α-topology, which is depicted as double lines and consists of three main tiers: (i) one front-end tier developed as a PHP application, one (ii) backend tier developed in Java and delivered as a WAR package, and a (iii) backend SQL database. The Web shop application μ-topology comprises an initial set T^{γ} containing five γ-topologies.

Each viable $T_j^{\mu}, j = 1, ..., i \ \epsilon \ T^{\mu}$ is constructed using the f_{dist} function. For example, a possible μ-topology for T_1^{α} can comprise the subset of γ-topologies τ^{γ}= $\{T_1^{\gamma}, T_3^{\gamma}, T_4^{\gamma}\}$.

As yet, we defined the functional aspects of the application. However, non-functional aspects, such as performance and cost, play a fundamental role in deciding among different cloud offerings. Therefore, let's denote the set of business and operational requirements for an application α-topology as $R = \{R_1, ..., R_j\}$. Each application requirement can be evaluated using specific measurements for each time interval in Ψ_m . Therefore, we define:

— the family of measurement sets $M = \{M_1, ..., M_v \mid v = 1, ..., j \ \wedge \ g_{map} : M \ \rightarrow \ R\}$, where g_{map} maps each measurement element set $M_v, v = 1, ..., j$, with its corresponding requirement in R, and
— the measurement sample $m = \{m_1, ..., m_s \mid s = 1, ..., m \ \wedge \ t_{map} : m \rightarrow \ \Psi \ \ \wedge \ m \ \epsilon \ M_v$, where t_{map} maps each measurement sample m with a time interval Ψ_m

For instance, for a requirement R_1 associated with the latency of the application, the set of measures $M_1 = \{\{1.5\,ms, ..., 2\,ms\}, ..., \{12.5\,req/s, ..., 9\,req/s\}\}$ would contain the average daily latencies for each time interval Ψ, e.g., for $\Psi_1 = \{Mo\ 18.7.2016, ..., Su\ 24.7.2016\}$.

The application workload is comprised by the probabilistic distribution of transactions that arrive over a time interval and performed by the different users of the application [Gómez Sáez et al. 2016]. Focusing on the workload behaviors that an application may receive over time, we define the set of application workload behaviors as $W = \{W_1, ..., W_k \mid W \sim D\}$. W is probabilistically distributed over each time interval $\Psi_m \ \epsilon \ \Psi$. For example, a Poisson distribution with $\lambda = 4$ may be used to describe the occurrence of a workload in a time interval Ψ. Then, $W \sim P(\lambda = 4)$.

3.2. Utility Function

The utility function, defined jointly by business and application architects, $u : T^{\mu} \times R \times M \times W \times \Psi_m \rightarrow \mathbb{R}$, for an application viable distribution T^{μ}, its set of requirements R and associated measures M, the set of workloads W, and its time interval $\Psi_m \ \epsilon \ \Psi$ is defined as:

$$u(T^{\mu}, R, M, W, \Psi_m) = revenue(T^{\mu}, M, W, \Psi_m) \ \times \ sat(\Psi_m, W) - cost(T^{\mu}, R, M, \Psi_m) \quad (1)$$

where $revenue : T^{\mu} \times M \times Wx \times \Psi_m \rightarrow \mathbb{R}$ is a function calculating the application's expected revenue during the time interval Ψ_m, and $cost : T^{\mu} \times M \times W \times \Psi_m \rightarrow \mathbb{R}$ is a function estimating the associated resources costs for the application viable topology T^{μ}. The *revenue* and *cost* functions are depicted in Sec. 3.3 and Sec. 3.4, respectively. The application's utility is influenced by the overall satisfaction of its end users, which can be calculated by the function $sat : \Psi \times W \rightarrow \mathbb{R}_{\geq 0}$, and impacts the total application's revenue. One possible definition of the $sat(\Psi, W)$ can be realized in terms of customer attrition:

$$sat(\Psi, W) = \frac{user_{gained}(\Psi, W)}{user_{loss}(\Psi, W)} \quad (2)$$

where $user_{gained} : \Psi \times W \rightarrow \mathbb{I}_{\geq 0}$ returns the average amount of end users gained in Ψ and $user_{loss} : \Psi \times W \rightarrow \mathbb{I}_{\geq 0}$ the average amount of turned over end users. For instance, for the workloads w_1 and w_2 during Ψ_2, $sat(\Psi_2, w_1) = 10$ and $sat(\Psi_2, w_2) = 2,5$, respectively.

As supported in SCARM, cloud-based applications can be redistributed. Therefore, the utility model must consider the impact on the utility when redistributing the application components, i.e. when adopting a new viable topology T^{μ}. Marginal utility is used to calculate the difference in utility when redistributing the application. The marginal utility for an application viable topology T^{μ}, given the application requirements R, its corresponding measurements M, and the workload behaviors W, is defined as:

$$\triangle\, u(T^{\mu}, \Psi) = u(T^{\mu}_{i}, R, M, W, \Psi_m) \;-\; u(T^{\mu}_{i-1}, R, M, W, \Psi_{m-1}) \;-\; cost_{red}(T^{\mu}_{i-1}, T^{\mu}_{i}) \quad (3)$$

where $cost_{red} : T^{\mu}_{i-1} \times T^{\mu}_{i} \rightarrow \mathbb{R}$ is a function calculating redistribution costs due to the transition from T^{μ}_{i-1} to T^{μ}_{i}. For instance, a redistribution of the web shop application depicted in Fig. 5 from a T^{μ}_{1} to a new T^{μ}_{2} in the first week of December 2016 – previous to the Christmas season– could entail the migration of the web shop's front-end to a cluster of two VMs, each Ubuntu-based *Amazon EC2 t2.large* using the *EBS gp2* elastic storage system for caching purposes. Therefore, redistributing the application would consist of provisioning T^{μ}_{2} during Ψ_2. The calculation of the marginal utility (see Eq. 3), therefore, entails the calculation of the redistribution costs $cost_{red}(T^{\mu}_{1}, T^{\mu}_{2})$ for migrating the front-end to an *AWS Beanstalk* container, such as the costs produced for a planned downtime, e.g., 2500 USD. Therefore, considering the new utility $u(T^{\mu}_{2}, ...) = 25000$ USD and the previous utility $u(T^{\mu}_{1}, ...) = 15000$ USD , and an infrastructure cost $cost(T^{\mu}_{2}, ...) = 5000$ USD, the marginal utility $\triangle\, u(T^{\mu}_{2}, \Psi_2) = 7500$ USD.

Focusing on calculating the utility for the lifetime of the application Ψ, denoted as multiple time intervals Ψ_m, we can define the utility of the application w.r.t. its life time as:

$$u(\Psi) = \sum_{j=1}^{|\Psi|} \beta_j \cdot u(T^{\mu}, R, M, W, \Psi_j) \quad (4)$$

where $\beta_j = [0, 1],\ \sum_j \beta_j = 1$, are the weights reflecting the preference over the time intervals Ψ in the life time of the application. For example, during a business year, there may be monthly periods where an application depicted by its topology T^{α} may be the associated with the primary line of business. In such a case, $\beta_j \approx 1$ for the corresponding months.

3.3. Revenue Function

So far, we presented the utility function, geared towards the profitability of cloud-based applications. For the remainder of this section, we reuse the viable application topology T^{μ}_{2} previously introduced for exemplification purposes. The first fundamental block building such a function is the *revenue* function, which determines the expected monetary revenue of an application distribution for a time interval $\Psi_m\ \epsilon\ \Psi$. More specifically, the *revenue* is defined as:

$$revenue(T^{\mu}, M, W, \Psi_m) = \sum_{w \epsilon W} P(w) \cdot [((1-\varepsilon)\ \cdot\ tpu(w)) \cdot |\Psi_m|\ \cdot\ rpu(\Psi_m)) \cdot users(w) \cdot av(T^{\mu}, \Psi_m, M)\ ] \quad (5)$$

where $P(w)$ is the probability for a workload behavior $w\ \epsilon\ W$ in the time interval Ψ_m. Lets assume a constant workload w_1 with $P(w_1) = 0.05$, and a periodical workload w_2 with $P(w_2) = 0.3$. The function $tpu : W \rightarrow \mathbb{R}_{\geq 0}$ depicts the average number of economic transactions per end user (customer) in a workload $w\ \epsilon\ W$, and ε represents the average transaction error rate of the application in the workload $w\ \epsilon\ W$. For example, for the workloads w_1 and w_2 previously depicted, the average number of transactions per end user may be 1 and 3 transactions, respectively, while the average transaction error rate of the application ε could be 0.03.

The function $rpu : \Psi \rightarrow \mathbb{R}$ is a given business function estimating the average monetary revenue per end user during the time interval in Ψ, which is typically developed by business architects and based on the analysis of seasonal monetary revenues. One possible example could be a step function depicting the average revenue per user per transaction on a monthly basis, returning an average of 80 USD per user for the months of December and January

(Christmas period), and 55 USD for the remainder months. The number of end users constituting a workload W is returned by the function $users : W \rightarrow \mathbb{R}_{\geq 0}$.

The availability of the underlying environment highly impacts the application's revenue. In this utility model, the availability function $av : T^{\mu} \times \Psi \times M \rightarrow [0, 1]$ returns the average proportional uptime in Ψ of the cloud services depicted in $\tau^{\mu} \, \epsilon \, T^{\mu}$. Considering the uptime described in the SLA of Amazon Web Services (AWS)[2], the $av(T_2^{\mu}, \Psi_2, ...)$ is 0.9995.

Based on the previous parameter values, the revenue for T_2^{μ} during the time interval Ψ_2, for the workloads w_1 and w_2 in W, for 3500 users during Ψ_2, for an average error rate of 0.03, and for a set of observed measures M can be calculated as follows:

$$\begin{aligned} revenue(T_2^{\mu}, M, W, \Psi_2) = 0.05 \cdot [((1 - 0.03) \ \cdot 1 \, tx/user) \cdot 2 \, months \cdot 80 \, USD) \cdot \\ 3500 \, user \cdot 0.9995 \ ] + \\ 0.3 \cdot [((1 - 0.03) \cdot \ 3 \, tx/user) \cdot 2 \, months \cdot 80 \, USD) \cdot \\ 3500 \, user \cdot 0.9995) \ ] \ = \\ 0.05 \cdot 540484 \ + \ 0.3 \cdot 1621452 \ = 27024.2 + 486435.6 \ \simeq \ 513000(USD) \end{aligned} \tag{6}$$

3.4. Cost Function

A second influence factor in the utility function is the resources utilization costs (see Eq. 1). Given a viable application distribution T^{μ}, the set of requirements R, and the application workload W for a time interval $\Psi_m \, \epsilon \, \Psi$ the cost function $cost(T^{\mu}, R, M, \Psi_m)$ is defined as:

$$cost(T^{\mu}, R, M, \Psi_m) = C_{fixed}(T^{\mu}, R, \Psi_m) \ + \ C_{variable}(T^{\mu}, R, M, \Psi_m) \tag{7}$$

where the function $C_{fixed} : T^{\mu} \times R \times \Psi \rightarrow \mathbb{I}_{\geq 0}$ returns the fixed costs for provisioning and maintaining the cloud services in T^{μ}, based on the set of requirements R and the time interval Ψ_m. $C_{fixed}(T^{\mu}, R, \Psi_m)$ is defined as:

$$C_{fixed}(T^{\mu}, R, \Psi) = C_{fixed}(g_{\gamma}(T^{\mu}), R, \Psi) = \sum_{\delta \, \epsilon \, \tau^{\gamma}} C_{provider}(\delta, R, \Psi) \tag{8}$$

where $g_{\gamma} : T^{\mu} \rightarrow \tau^{\gamma}$ returns the set $\tau^{\gamma} \ \subseteq \ T^{\gamma}$ of γ topologies in a T^{μ}, and $C_{provider} : \delta \times R \times \Psi \rightarrow \mathbb{R}_{\geq 0}$ calculates the provisioning and maintenance costs for every sub-topology $\tau_q^{\gamma} \, \epsilon \, \tau^{\gamma}$. Focusing on the Web shop application example in Fig. 5 distributed w.r.t. T_2^{μ} during the time interval Ψ_2:

$$\begin{aligned} C_{fixed}(g_{\gamma}(T_2^{\mu}), R, \Psi_2) = C_{fixed}(\{T_1^{\gamma}, T_3^{\gamma}, T_4^{\gamma}\}, R, \Psi_2) = \sum_{\delta \, \epsilon \, \tau^{\gamma}} C_{provider}(\delta, R, \Psi_2) = \\ C_{provider}(T_1^{\gamma}, R, \Psi_2) + C_{provider}(T_3^{\gamma}, R, \Psi_2) + C_{provider}(T_4^{\gamma}, R, \Psi_2) \end{aligned} \tag{9}$$

However, the fulfillment of the set of requirements R depends on (i) the performance offered by a cloud provider, and on (ii) the application workload behavior. Therefore, we must also consider incurred costs due to the scaling of resources to satisfy every requirement in the set R, e.g., scaling out a virtual machine to satisfy a workload peak interval. Let's define the function calculating the variable costs $C_{variable} : T^{\mu} \times R \times M \times \Psi \rightarrow \mathbb{R}_{\geq 0}$ are defined as:

$$C_{variable}(T^{\mu}, R, M, \Psi_m) = C_{variable}(g(T^{\mu}), R, M, \Psi_m) = \sum_{\delta \, \epsilon \, \tau^{\mu}} \sum_{r \, \epsilon \, R} C_{adapt}(\delta, r, M, \Psi) \tag{10}$$

[2]Amazon Web Services (AWS) SLA: https://aws.amazon.com/ec2/sla/

where $C_{adapt}(\tau^{\mu}, R, M, \Psi)$ returns the sum of incurred scaling (additional) costs in τ^{μ} during the time interval Ψ to satisfy every requirement in R evaluated through each measure in M. In line with the previous example (see Fig. 5), a potential adaptation cost to satisfy a requirement $R_1 = database_{throughput} \geq 10$ *reqs./sec.* could consist of provisioning an additional VM instance *t2.medium* in τ_4^{γ} during a time interval Ψ_2. More specifically, one possible definition of c_{adapt} can be:

$$C_{adapt}(\delta, R, M, \Psi) = \begin{cases} C_{provider}(\delta, R, \Psi) & if\ ev(R, M, \Psi) \\ 0 & otherwise \end{cases}$$

where $ev : R \times M \times \Psi \rightarrow \mathbb{B}$ is a function that evaluates the fulfillment of a requirement in R for a time interval Ψ and w.r.t. its measure sample in M. The previously introduced utility model can be leveraged in the cost- and performance-efficient distribution of applications. The remainder of this work focuses on evaluating the utility model under different distribution scenarios.

4. EVALUATION: MEDIAWIKI CASE STUDY

The MediaWiki[3] application serves as the underlying technology supporting the Wikipedia[4] project, part of the Wikimedia Foundation (WMF)[5]. MediaWiki fundamentally aggregates two tiers: a (i) front-end PHP application comprising the business logic and caching functionalities, and a (ii) backend SQL database (see Fig. 1 in Sec. 2.1). It typically serves several million of users on a daily basis, and its revenue model is primarily based on donations. The WMF identified in its Y2016 Annual Plan[6] a set of potential risks for the Foundation's mission, among which are the following: (i) failure in technology infrastructure cause a disruption of WMF operations, and (ii) efforts to build large scale, high performance features result in delays or failures. Migrating MediaWiki to a cloud environment opens a wide set of possibilities and challenges, as it can leverage the on-demand usage of resources, high availability, and a reduction of maintenance and management efforts of on-premise infrastructure resources. In the remainder of this section we present the conceptual and empirical evaluation of the previously presented utility model by applying it to Wikipedia.

4.1. Methodology & Setup

The evaluation focuses on (i) generating a first set of viable topologies (T^{μ}) of MediaWiki, by applying the *Discovery & Evaluation* step of SCARM (see Fig. 4 in Sec. 2), and on (ii) evaluating the different T^{μ} by ranking the viable topologies using different utility models.

4.1.1. Application Viable Distributions. The generated viable application distributions are depicted in Table I, comprising VM- and container-based services from two major cloud providers, AWS[7] and Microsoft Azure[8]. This first set of experiments are scoped to a single-provider distribution of applications, i.e., hosting the whole MediaWiki stack within one provider, but among different (types of) services. Future experimental rounds are planned to evaluate multi-cloud distributions and the impact of performing a redistribution of the application tiers.

Towards minimizing network latency to both AWS and Azure infrastructures, we utilized AWS and Azure EU regions, both provisioned in Ireland. Table I depicts the set of evaluated distributions $\{T_1^{\mu}, ..., T_8^{\mu}\}$. These entail the usage of VM- and containerized-based

[3]MediaWiki: https://www.mediawiki.org
[4]Wikipedia Project: https://www.wikipedia.org
[5]Wikimedia Foundation: https://wikimediafoundation.org
[6]WMF Y2016: https://upload.wikimedia.org/wikipedia/foundation/4/43/WMF2015-16AnnualPlan.pdf
[7]AWS: https://aws.amazon.com
[8]Microsoft Azure: https://azure.microsoft.com/en-us/

Table I: Evaluation Setup - Viable Distributions (T^{μ}) of the MediaWiki Application. *Prices are calculated for the on-demand usage. Storage and data transfer costs are billed separately.*

T^{μ}	**Service**	**MediaWiki Front-end**	**MediaWiki Back-end**	**Auto-Scaling**	**Region**	**Total Price (USD/h)**
T_1^{μ}	EC2	m4.large	m4.large	✗	EU (IR)	0.234
T_2^{μ}	EC2	m4.xlarge		✗	EU (IR)	0.264
T_3^{μ}	EC2 + RDS	m4.large	db.m4.large	✗	EU (IR)	0.325
T_4^{μ}	Beanstalk	(2x) t2.small	db.m4.large	✓	EU (IR)	(2x) 0.028 + 0.193
T_5^{μ}	ECS	(2x) t2.small	db.m4.large	✓	EU (IR)	(2x) 0.028 + 0.193
T_6^{μ}	VM	DS2	DS2	✗	EU (IR)	0.292
T_7^{μ}	VM	DS3		✗	EU (IR)	0.292
T_8^{μ}	Container	(2x) DS1	DS2	✓	EU (IR)	(2x) 0.073 + 0.146

☐ Amazon Web Services ☐ Windows Azure Services

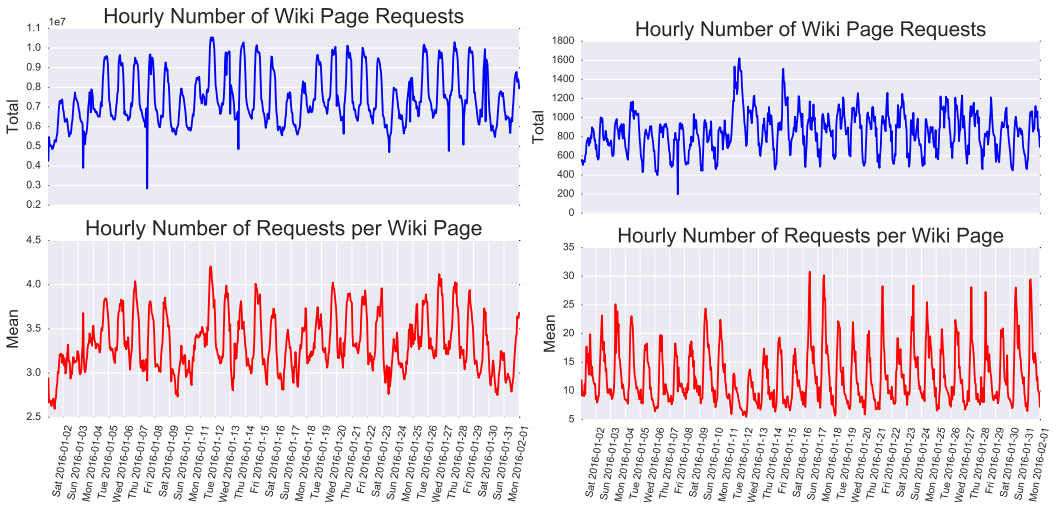
environments, with and without auto-scaling capabilities. W.r.t. the autoscaling, the cluster consisted of a minimum of one and a maximum of two VM instances, with the scaling trigger configured when the CPU usage exceeds 50%. In the scenarios utilizing containerized environments, we developed a set of Docker images available in GitHub[9]. Due to the incompatibility of MediaWiki[10] with the SQL servers offered in Microsoft Azure, we cannot consider the deployment of the MediaWiki back-end database in an Azure DBaaS offering.

The prices depicted in Table I are the total hourly prices for the front- and back-end tier stacks on the different services. Storage, load balancer instances, and network egress charges are calculated separately, as these depend on the application profile and are influenced by the auto-scaling mechanism. Therefore, these are part of the fixed and variable costs $C_{fixed}(T_i^{\mu}, ...)$ and $C_{variable}(T_i^{\mu}, ...)$, respectively. The availability function is extracted from the SLAs provided by the cloud providers. More specifically, the availability function $av(T^{\mu}, \Psi_m, M)$ is defined as a constant function, which returns the availability defined in the cloud provider's SLA for each T_i^{μ}.

4.1.2. Workload Analysis, Generation & Execution. A second step consists of analyzing and generating a system load that emulates the usage of MediaWiki. Wikidumps[11] provides real access traces, number of users, and databases dumps of Wikipedia. Therefore, we leverage such data to emulate the load in our system. In particular, since we don't aim at hosting a complete Wikipedia mirror site, but to evaluate the defined concepts and models using a realistic workload behavior and data as the basis, we generated a scaled version of such a workload. We selected the English Wikipedia pages data, access traces, and users access, for the time interval comprised from Jan. 1 2016 to Jan. 31 2016. The English Wikipedia represents ≈ 46% of all Wikimedia projects, and received the highest load (7,869M views) in January 2016, showing a 9% increase w.r.t. the previous year, according to Wikistats[12].

Figure 6 depicts the hourly access traces of the English Wikipedia for January and the generated access traces for the experiments. The workload generation consists of: (i) cleaning the English Wikipedia access traces (e.g. error requests), (ii) scaling the number of requests to Wiki pages using a 1000:1 factor, and (iii) verifying that the generated workload still correlates users and requests in a similar manner w.r.t. the original. Correlation showed a minor decrease of the pearson correlation among requests and users in the generated

[9]MediaWiki Docker: https://github.com/sgomezsaez/SCARF-Evaluation/tree/master/mediawiki_docker
[10]MediaWiki Compatibility: https://www.mediawiki.org/wiki/Compatibility
[11]Wikidumps: https://dumps.wikimedia.org/
[12]https://stats.wikimedia.org/EN/TablesPageViewsMonthlyCombined.htm



(a) Hourly Summary Requests - Original

(b) Hourly Summary Requests - Generated Workload Sample

Fig. 6: Wikipedia Workload Analysis January 2016

workload of ≈ 0.05. The workload was generated using Python 2.7 SciPy[13], and comprises a total of 632K requests and 79K users.

The load execution and the processing of results are driven in Apache JMeter 2.9[14] and Python scripts, respectively. The workload generation scripts, generated load, as well as the load profile used in JMeter, are available in GitHub[15].

4.1.3. Revenue per User & Satisfaction Model. A third step in the evaluation consists of calculating the average revenue per user, which is typically derived by business architects for the revenue model (see Sec. 3.3). Revenue models are built based on past data and are used in order to predict and establish business revenues and objectives, respectively. We analyzed the English Wikipedia revenue in the WMF Fundraising Data[16], by means of processing the total daily income, which is based on donations (see Fig. 7). The WMF regularly organizes campaigns that aim at significantly increasing the number of donations during a time period, such as the one driven in December 2015[17] for the English Wikipedia, which consisted of adding a banner that redirects to their donations platform. The peak depicted at the beginning of January 2015 is due to the residuum of such campaign, i.e., users which donated after the campaign ended. The revenue for January 2016 follows an exponential function, as described in Fig. 7, and the average revenue fit is 0.000534 USD/user. However, such a function may only fit for January 2016, and not for other months.

W.r.t. the satisfaction model, we followed in Eq. 11 a similar user survival analysis heuristic as in the WMF Measuring User Search Satisfaction Schema 2.0.0[18]. We defined the maximum waiting time for a user to retrieve a Wiki page to 30s. Beyond such interval, we consider the

[13]SciPy: https://www.scipy.org
[14]Apache JMeter: http://jmeter.apache.org
[15]SCARF-Evaluation: https://github.com/sgomezsaez/SCARF-Evaluation/
[16]WMF Fundraising Data: https://frdata.wikimedia.org
[17]https://meta.wikimedia.org/wiki/Fundraising#December_2015_Campaign_Launch_Update
[18]Measuring User Search Satisfaction Schema 2.0.0: https://meta.wikimedia.org/wiki/Research:Measuring_User_Search_Satisfaction

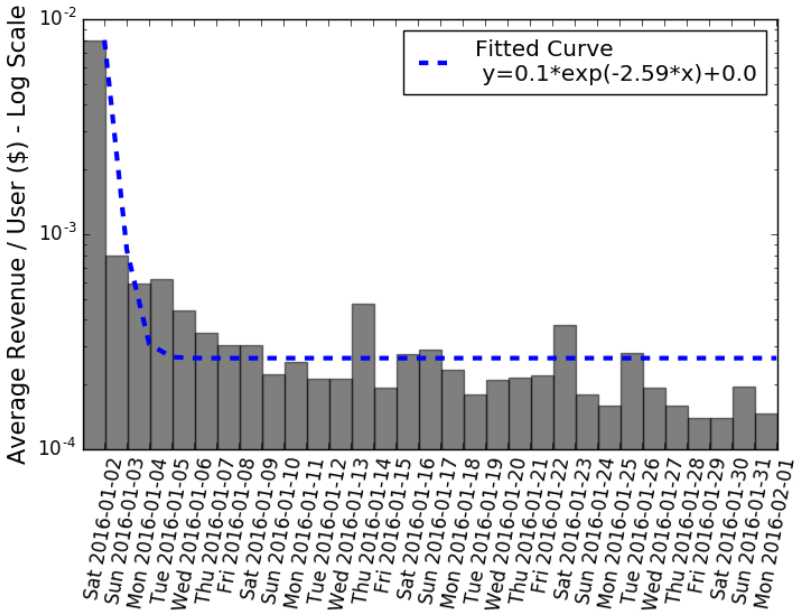


Fig. 7: Wikimedia Foundation Revenue - January 2016

user to abandon the application. Besides, we also considered the total number of unsuccessful requests, i.e., requests with an HTTP return code 50x. The satisfaction is defined as:

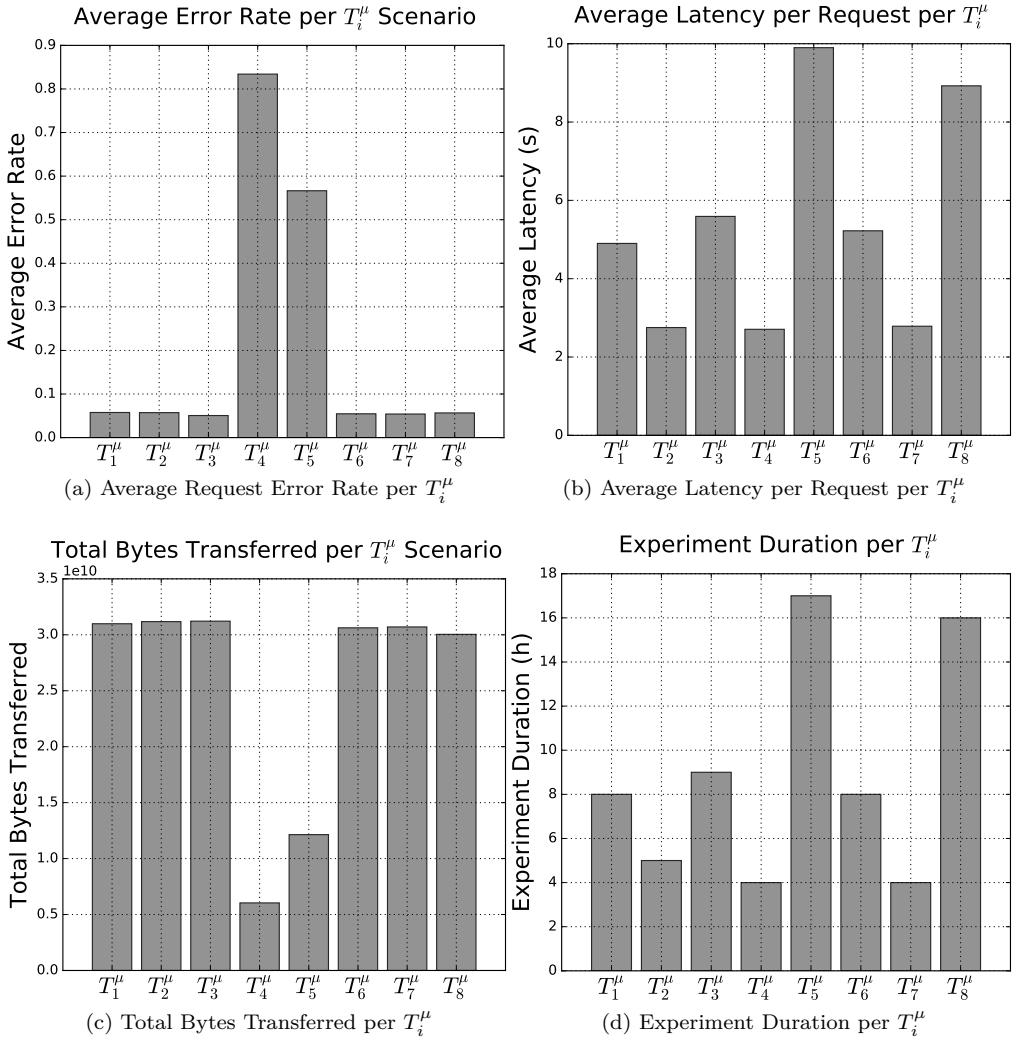
$$sat(\Psi, W) = 1 - \frac{reqs_{>30s}(\Psi, W) + reqs_{error}(\Psi, W)}{reqs_{total}(\Psi, W)} \tag{11}$$

where $reqs_{>30s}(\Psi, W) : \Psi \times W \rightarrow \mathbb{I}_{\geq 0}$ returns the total number of requests in the workload W with a latency greater than 30s, $reqs_{error}(\Psi, W) : \Psi \times W \rightarrow \mathbb{I}_{\geq 0}$ returns the total number of unsuccessful requests in the workload W, and $reqs_{total}(\Psi, W) : \Psi \times W \rightarrow \mathbb{I}_{\geq 0}$ returns the total number of requests in the workload W.

4.1.4. Utility Calculation & Comparison. The last step in the evaluation analyzes the utility for each T^{μ} viable distribution using Eq. 1. We first derived the average revenue per user in Eq. 5 and calculated the expected infrastructure costs (see Eq. 7) for the different utilized services. Subsequently, we calculated the utility of the different MediaWiki T^{μ} viable distributions and ranked them accordingly. As utility can be calculated with different functions, we compared our approach (see Eq. 1) with further functions (see Eq. 12), considering only infrastructure costs, as in [Andrikopoulos et al. 2014], or the cloud provider's availability:

$$\begin{aligned} u_{opex}(T^{\mu}, R, M, W, \Psi_m) &= opex_{max} - opex(T^{\mu}, R, M, \Psi_m) \\ u_{av}(T^{\mu}, R, M, W, \Psi_m) &= av(T^{\mu}, \Psi_m, M) \end{aligned} \tag{12}$$

where $opex_{max}$ represents the maximum operational cost, and $av(T^{\mu}, \Psi_m, M)$ returns the average expected uptime for the providers involved in T_i^{μ}. The outputs of such calculations are ranked and subsequently plotted using Python 2.7 Pyplot libraries.



(a) Average Request Error Rate per T_i^μ

(b) Average Latency per Request per T_i^μ

(c) Total Bytes Transferred per T_i^μ

(d) Experiment Duration per T_i^μ

Fig. 8: Experiment Results for the T_i^μ Alternative Distributions

4.2. Experimental Findings

This section tailors the experimental findings into (i) a performance analysis focusing on metrics relevant to the utility model previously introduced, and (ii) the utility calculation and ranking of the different T_i^μ viable distributions, using different utility functions.

4.2.1. Experimental Results. Fig. 8 details the observed performance latency of the different T_i^μ viable distributions. In particular, when analyzing the average request error rate for each T_i^μ, we observe that the error rate maintains steady ($\lesssim 6\%$) among the viable distributions $T_1^\mu, T_2^\mu, T_3^\mu, T_6^\mu, T_7^\mu$, and T_8^μ, observing the higher error rates when consuming AWS services. In contrast, T_4^μ and T_5^μshow approximately a 90% increase of the error rate w.r.t. the minimum observed (≈ 0.056 for T_3^μ). Such a large difference is due to the overload observed in both the AWS load balancer and the VM instances, which returned for most of the requests with *Service Unavailable* or *Gateway Timeout* errors, and consumed in average

$\approx 90 - 100\%$ of the CPUs. Such a high consumption impacted the auto-scaling mechanism, which provisioned permanently 2 VM instances in the cluster. A T_i^μ application distribution with a high error rate has a negative impact on the utility, and on the position of such a T_i^μ in the ranking. SCARF does not discard viable distributions, but ranks them according to its utility. This ranking serves as a guidance for application architects for selecting a viable distribution of their application.

The average hourly latency in Fig. 8b depicts a performance increase of $\approx 45\%$ when deploying the full MediaWiki stack in a VM, in contrast to the scenarios hosting the front-end and back-end in separate VMs. When using a DBaaS to host the back-end database (T_3^μ), there exists a minor performance decrease of $\approx 10\%$ w.r.t. distributing both the front- and back-end in separate VMs. However, there exists a fair decrease of management tasks and their corresponding costs, due to the fact that DBaaS offerings transparently administer and manage database servers. The MediaWiki distribution in the Azure Container Service shows increased latency of $\approx 40\%$ when distributing both front-end and back-end in separate Azure VMs. Focusing on the cloud providers, AWS offers on average $\approx 6\%$ performance increase for the scenarios distributing the MediaWiki front- and back-end in separate VMs, and $\approx 1\%$ for the scenarios hosting the whole MediaWiki stack in a VM.

W.r.t. the data transfer for each scenario (see Fig. 8c), we can observe that for the scenarios comprising the $T_1^\mu, T_2^\mu, T_3^\mu, T_6^\mu, T_7^\mu$, and T_8^μ viable distributions, a total of $\approx 29GB$ are transferred between the load driver and the MediaWiki application. However, such amount of data transferred significantly decreases in the viable distributions T_4^μ and T_5^μ, since the majority of the responses consist of error codes and messages, rather than the requested Wiki pages. The total experiment duration for each T_i^μ is depicted in Fig. 8d and conforms with the observed latency. The viable distributions T_5^μ and T_8^μ consumed most of the time in comparison to the remainder T_i^μ viable distributions.

Since the MediaWiki is a read-intensive application, i.e., heavily searches and retrieves Wiki content, we observe a performance degradation when distributing the MediaWiki front- and back-ends in separate VMs or DBaaS services. However, such a deviation is not significant if we consider the benefits of using specialized services, such as DBaaS offerings, in the sense that administration and management costs of the back-end database significantly decrease. When using container clusters, a performance degradation was also observed during runtime. Considering and measuring the provisioning and deployment time of containers may, however, overturn such results, and are planned as future work by means of analyzing the utility when redistributing the application. In the following section, we utilize the experimental results to evaluate the different T_i^μ using the utility models previously introduced, considering the (i) application profitability, (ii) operational costs, and (iii) cloud infrastructure availability.

4.2.2. Utility Ranking & Analysis. The usage of utility as the underlying model for decision making can considerably assist application and business architects to prosper in the distribution of their applications among cloud services and providers. In the context of this work, we utilize the utility models to evaluate and rank the T_i^μ viable distributions. More specifically, we first apply the utility function depicted in Eq. 1, and compare it with further utility functions, which only consider the operational expenses or the cloud provider's availability (see Eq. 12). The utility results are depicted in Fig. 9 and the ranking of T_i^μ viable distributions consists of an ordered preference structure built upon the calculation of the different T_i^μ viable distributions (see Eq. 13). When utilizing the utility function depicted in this work and used in SCARF, results show that the optimal viable distribution is T_7^μ, which returns $\approx 480USD$ (US Dollars) for the evaluated period. Due to the low performance and high error rate observed in the T_4^μ and T_5^μ viable distributions, such viable distributions offer the lowest utility, $\approx 60USD$ and $\approx 1USD$, respectively.

When using the utility function that exclusively focuses on the expected operational costs (see Eq. 12), we see that the optimal viable distributions are T_7^μ, T_6^μ, and T_8^μ. This

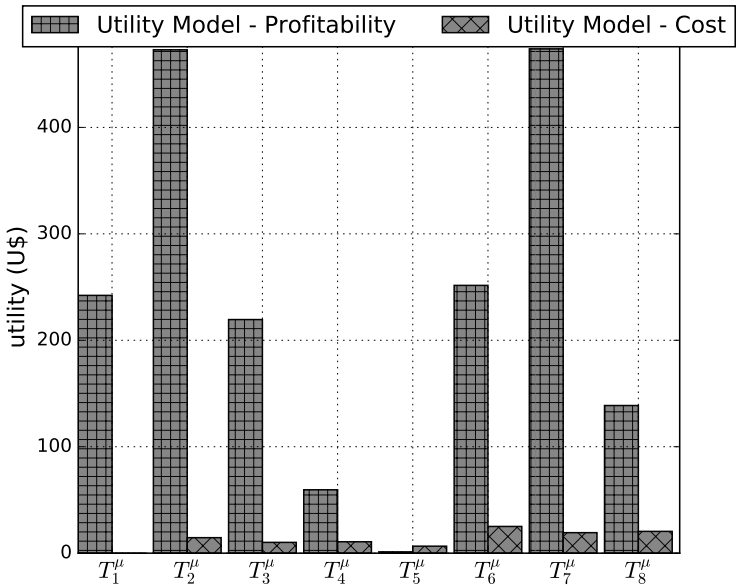


Fig. 9: Experimental Results - Utility Analysis & Comparison

is due to lower incurred storage and network transfer costs offered in Azure. Since only the infrastructure costs are considered, viable distributions like T_2^μ are not considered as favorable distributions, which in practice offer a better performance. Moreover, T_4^μ is not ranked as a less favorable distribution, therefore negatively impacting in the selection of a viable T_i^μ distribution due to its low performance and high error rate. If we consider the utility function that exclusively focuses on the availability, as described in Eq. 13, we cannot establish a strict preference order, since both AWS and Azure express a guaranteed 99.95% uptime availability for VMs provisioned in the same availability zone. The lack of performance knowledge in the utility model can, therefore, negatively impact in the decision making tasks, as viable distributions offering a low operational costs may provide a degraded performance and, therefore, a low monetary return.

$$\begin{aligned} \textit{SCARF Utility} &: T_7^\mu \succ T_2^\mu \succ T_6^\mu \succ T_1^\mu \succ T_3^\mu \succ T_8^\mu \succ T_4^\mu \succ T_5^\mu \\ \textit{Cost} &: T_7^\mu \succ T_6^\mu \succ T_8^\mu \succ T_2^\mu \succ T_1^\mu \succ T_4^\mu \succ T_3^\mu \succ T_5^\mu \\ \textit{Availability} &: T_1^\mu \succeq T_2^\mu \succeq T_3^\mu \succeq T_4^\mu \succeq T_5^\mu \succeq T_6^\mu \succeq T_7^\mu \succeq T_8^\mu \end{aligned} \tag{13}$$

Focusing on the sensitivity of the SCARF utility model, it is mainly influenced by the parameters of its three fundamental functions $revenue(T^\mu, M, W, \Psi_m)$, $sat(\Psi_m, W)$ and $cost(T^\mu, R, M, \Psi_m)$ depicted in Sec. 3.1. The revenue function $revenue(T^\mu, M, W, \Psi_m)$ is mainly impacted by the (i) application distribution, the (ii) observed performance, and the (iii) business revenue during Ψ_m. Focusing on the former, an application distribution T^μ offering a higher availability and lower error rate ε increases the application's revenue, therefore increasing the application's utility. A low performance negatively impacts the revenue of the application, as it is capable of serving a lower amount of transactions per user $tpu(w)$ for the each workload w. Lastly, a higher business revenue per user $rpu(\Psi_m)$ elicits a higher overall application revenue, therefore increasing the application's distribution utility.

The end user satisfaction $sat(\Psi_m, W)$ proportionally impacts the application's revenue, in the sense that a higher end user satisfaction increases the application's utility. The satisfaction function is domain-specific. For instance, the MediaWiki satisfaction Function 11

sets a threshold of 30s for successful requests. A lower threshold would imply a more rigid computation of the satisfaction, by triggering a decrease of the end user's satisfaction, and a reduction of the overall application's utility. Lastly, the cost function $cost(T^{\mu}, R, M, \Psi_m)$ negatively impacts the application's utility. In particular, a higher application distribution cost lowers the application utility. Since the application's cost depends on the $C_{fixed}(T^{\mu}, ...)$ and $C_{variable}(T^{\mu}...)$, the variation of any of its parameters negatively or positively impact the application's utility, if such costs increase or decrease, respectively.

5. DISCUSSION

The previous sections presented and evaluated the utility model in SCARF. Such a model allows to rank viable cloud-based application distributions, depicted as viable topologies T_i^{μ}, using utility theory as the basis. In summary, the utility model (i) allows application and business architects to compute and analyze the trade-off between cost and performance when hosting their applications in the cloud, and (ii) serves as a mechanism to rank alternative viable topologies w.r.t. business and operational performance and cost requirements.

The previous empirical evaluation demonstrated the benefit of using the proposed utility model, in comparison with only considering operational costs or availability as decision support for selecting cloud services. Although the current utility model can be leveraged for efficiently distributing applications in the cloud, there are limitations to both the model itself and its evaluation: (i) cost calculation in SCARF's utility function is currently limited to linear cost models, such as hourly utilization, or reserved instances. More complex cost models, such as spot instances, are not yet supported in SCARF, as these require more complex analytical models. W.r.t. the experiments, scenarios only consider single-cloud distributions of applications, i.e., deploying the whole application stack in one cloud provider. We are aware of such a limited scope in this first set of experiments, and plan to go a step further, e.g., (i) focusing on multi-cloud distribution scenarios, (ii) evaluating the effectiveness of application redistributions, and (iii) providing a cookbook on the necessary steps for migrating such applications to the cloud.

6. RELATED WORKS

Optimization of QoS in the cloud can be organized into two main categories: (i) resources management and allocation in cloud environments, and (ii) design support and optimization frameworks for migrating applications to the cloud. Bellow the first category, utility is used (i) in management of cloud resources or (ii) in autonomic computing. Utility-based techniques are typically used in IaaS environments for scheduling VMs. Minarolli and Freisleben analyze the trade-off between QoS and operational costs and look into maximizing a global utility in a IaaS environments [Minarolli and Freisleben 2011b]. The authors consider CPU allocation and costs per CPU, and calculate the utility when allocating virtual machines in physical nodes. Similarly, Goudarzi and Pedram [Goudarzi and Pedram 2011] analyze the user's observed response, and focus on optimizing the processing, memory, communication resources, and service levels. Focusing on maximizing revenue in IaaS environments, Hong and Baochun propose an infrastructure revenue model based on dynamic pricing [Xu and Li 2013]. In storage services, Strunk et al. use utility functions to provision and maintain distributed storage systems [Strunk et al. 2008]. However, the previous approaches focus on independently optimizing concrete types of cloud services, such as VMs or storage services.

In design and decision support systems for migrating applications to the cloud, model transformation and simulation techniques are common approaches. CloudMig [Frey and Hasselbring 2011] builds on an initial topology and utilization model that is transformed to optimize the configuration of VM resources in the application model. The evaluation and comparison of generated architectures is not yet supported in CloudMig. Similarly, the MODACloud [di Nitto et al. 2013] and CloudML [Brandtzæg et al. 2012] approaches focus on providing a multi-dimensional early design support of applications by applying

model transformation techniques and code generation for multi-cloud applications. However, empirical evaluations for these approaches focus on provisioning VMs in a multi-cloud environment, rather than considering further application deployment approaches, such as container-based. The SeaClouds EU Project[19] provides a *Cloud Service Orchestrator* capable of provisioning and managing application components spanned among multiple Cloud environments [Brogi et al. 2014]. The CACTOS EU Project[20] is possibly the closest approach to the fundamentals developed as part of this work. as it fits provider resources for diverse application workloads. Focusing on explicitly comparing cloud providers, the CloudCmp framework provides assessment in terms of elasticity, persistent storage, and networking services in different cloud providers, and works towards creating an end-to-end benchmark for provider's optimization [Li et al. 2010]. The previous approaches present the following drawbacks. Firstly, these introduce complex tasks, such as the creation of simulation models, which often require the intervention of domain experts. These cause an overhead in the development and (re)deployment tasks of applications. Secondly, they mostly focus on running applications hosted on VM environments, rather than considering the performance and cost of further deployment options, such as containerized environments. Therefore, these do not fully exploit the usage of cloud services to evaluate and optimize the profitability of cloud-based applications.

Works in the domain of optimization of cloud-based application viable distributions mostly focus on estimating operational expenses and QoS objectives. Miglierina et al. aim at optimizing availability and operational expenses [Miglierina et al. 2013], by means of defining a Palladio-based application topology model, which is used to simulate the application performance. The MOCCA framework [Leymann et al. 2011] introduces variability points in the application topology to cope with possible alternative deployments. CMotion [Binz et al. 2011] utilizes multiple criteria defined by domain experts to analyze and generate application topologies. Approaches like Kingfisher [Sharma et al. 2011], CloudGenius [Menzel and Ranjan 2012], CloudAdoption [Khajeh-Hosseini et al. 2012] focus on optimally selecting cloud offerings for monolithic applications based on infrastructure costs. These approaches are therefore limited in their usefulness w.r.t. the existence of multiple viable and potential application distribution alternatives across different types of cloud offerings. Nevertheless, all of the above approaches assume that the application topology is already known (and fixed), and are restricted to VM-based solutions. Our approach takes into account also non-VM cloud services like DBaaS offerings, and allows for the dynamic generation of viable application topologies by reusing, discovering, and selecting γ-topology for the application. The usage of multi-objective genetic algorithms to narrow the space of available offerings is investigated by Amato and Venticinque [Amato and Venticinque 2016]. However, it exclusively focus on SLA conditions and lack of a method to select a concrete cloud offering for the application. Ye et al. introduce an economic model for composing in IaaS services [Ye et al. 2014]. Our work goes a step further by generalizing and eliminating constraints w.r.t. the cloud service used to distribute the application.

The vision explored in this paper goes beyond the state of the art by (i) fostering the collaboration of business and IT experts towards a common model for distributing cloud-based applications in a profitable manner, (ii) providing concepts and techniques for facilitating the exploration and decision making tasks for distributing cloud-based applications among different types of cloud services, and by (iii) empirically evaluating the usage of utility to rank the distribution of a realistic application and workload using VM, containerized, and database cloud services.

[19]SeaClouds EU Project: http://www.seaclouds-project.eu/project.html
[20]Cactos EU Project: http://www.cactosfp7.eu/

7. CONCLUSIONS

Migrating applications to the cloud has become in the last years a challenging task, due to the (i) wide amount of cloud services and providers, and the (ii) heterogeneity of cloud offerings in terms of their cost and performance. Due to such complexity, there exists a necessity to develop decision making concepts and techniques to assist application and business architects to optimally select and configure cloud services and providers to distribute their applications.

This work leverages utility theory to evaluate during design phase the profitability of applications spanned among multiple cloud services. We introduced a life-cycle and method that (i) evaluates viable cloud-based application distributions, and (ii) ranks such viable distributions to assist application and business architects to efficiently select cloud services. In particular, the utility model evaluates a viable application distribution from an economic perspective, i.e. its expected monetary return by computing performance and resources costs, availability, and end user satisfaction when consuming cloud services. The evaluation of the utility model is driven using the MediaWiki (Wikipedia) application as case study, its realistic workload, and its publicly available financial data. In particular, we applied the utility model to evaluate multiple viable topologies of MediaWiki, each using different cloud services, such as AWS EC2 and ECS, RDS, Beanstalk, Azure VM, and Azure Container Service. Experimental results show the benefit when using the proposed utility model for deciding among different application distributions in the cloud, in comparison with considering cost or availability independently. Such a benefit relies on the fact that the proposed utility model considers not only availability and cost, but also application's workload and revenue model jointly. Future works focus on continuing the evaluations, by (i) driving an extended sensitivity analysis on the model, and by evaluating (ii) multi-cloud deployment and (iii) redistribution scenarios. Moreover, further types of application architectures, e.g., workflow-based or built as micro-services are planned to be considered. Ongoing works are also aligned with incorporating the analysis of complex pricing models, e.g., *spot instances* in AWS.

ACKNOWLEDGMENTS

The authors thank Florian Frech and Maria Elena Alonso Mencia for their contributions.

REFERENCES

Alba Amato and Salvatore Venticinque. 2016. Multiobjective Optimization for Brokering of Multicloud Service Composition. *ACM Trans. Internet Technol.* 16, 2, Article 13 (April 2016), 20 pages.

Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. 2013. How to Adapt Applications for the Cloud Environment. *Computing* 95, 6 (2013), 493–535.

Vasilios Andrikopoulos, Santiago Gómez Sáez, Frank Leymann, and Johannes Wettinger. 2014. Optimal Distribution of Applications in the Cloud. In *Proceedings of CAiSE'14*. Springer, Springer, 75–90.

Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.

Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak, and Sebastian Wagner. 2013. OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of ICSOC'13 (LNCS)*, Vol. 8274. Springer Berlin Heidelberg, 692–695.

Tobias Binz, Frank Leymann, and David Schumm. 2011. CMotion: A Framework for Migration of Applications into and between Clouds. In *Proceedings of SOCA'11*. IEEE Computer Society, 1–4.

Eirik Brandtzæg, Parastoo Mohagheghi, and Sébastien Mosser. 2012. Towards a domain-specific language to deploy applications in the clouds. In *Proceedings of CLOUD COMPUTING'12*. IARIA, 213–218.

Antonio Brogi, Ahmad Ibrahim, Jacopo Soldani, José Carrasco, Javier Cubo, Ernesto Pimentel, and Francesco D'Andria. 2014. SeaClouds: a European project on seamless management of multi-cloud applications. *ACM SIGSOFT Software Engineering Notes* 39, 1 (2014), 1–4.

Cloud Standards Customer Council. 2013. Migrating Applications to Public Cloud Services: Roadmap for Success. (December 2013).

Elisabetta di Nitto, Marcos Aurélio Almeida da Silva, Danilo Ardagna, Giuliano Casale, Ciprian Dorin Craciun, Nicolas Ferry, Victor Muntes, and Arnor Solberg. 2013. Supporting the development and

operation of multi-cloud applications: The ModaClouds approach. In *Proceedings of SYNASC'13*. IEEE, 417–423.

M. Fowler. 2002. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Sören Frey and Wilhelm Hasselbring. 2011. The cloudmig approach: Model-based migration of software systems to cloud-optimized applications. *International Journal on Advances in Software* 4, 3 and 4 (2011), 342–353.

Santiago Gómez Sáez, Vasilios Andrikopoulos, Michael Hahn, Dimka Karastoyanova, Frank Leymann, Marigianna Skouradaki, and Karolina Vukojevic-Haupt. 2015. Performance and Cost Evaluation for the Migration of a Scientific Workflow Infrastructure to the Cloud. In *Proceedings of CLOSER'15*. SciTePress, 352–361.

Santiago Gómez Sáez, Vasilios Andrikopoulos, and Frank Leymann. 2016. Consolidation of Performance and Workload Models in Evolving Cloud Application Topologies. In *Proceedings of CLOSER'16*. SciTePress, Rome, Italy, 160–169.

Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, and Steve Strauch. 2014. Design Support for Performance Aware Dynamic Application (Re-)Distribution in the Cloud. *IEEE Transactions on Services Computing* 8, 2 (December 2014), 225–239.

Hadi Goudarzi and Massoud Pedram. 2011. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *Proceedings of CLOUD'11*. IEEE, 324–331.

Rolf Harms and Michael Yamartino. 2010. The economics of the cloud. *Microsoft whitepaper, Microsoft Corporation* (2010).

Jez Humble and Joanne Molesky. 2011. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal* 24, 8 (2011), 6.

Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. 2013. Cloud migration research: a systematic review. *IEEE Transactions on Cloud Computing* 1, 2 (2013), 142–157.

Ralph L Keeney and Howard Raiffa. 1993. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press.

Ali Khajeh-Hosseini, David Greenwood, James W. Smith, and Ian Sommerville. 2012. The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience* 42, 4 (2012), 447–465.

Joseph Packy Laverty, David F Wood, and John Turchek. 2014. Micro and Macro Economic Analysis of Cloud Computing. *Issues in Information Systems* 15, 2 (2014).

Frank Leymann, Christoph Fehling, Ralph Mietzner, Alexander Nowak, and Schahram Dustdar. 2011. Moving applications to the cloud: An approach based on application model enrichment. *International Journal of Cooperative Information Systems* 20, 03 (2011), 307–356.

Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of IMC'10*. ACM, 1–14.

Alfred Marshall. 2009. *Principles of economics: unabridged eighth edition*. Cosimo, Inc.

Peter Mell and Tim Grance. 2011. The NIST definition of cloud computing. (2011).

Michael Menzel and Rajiv Ranjan. 2012. CloudGenius: decision support for web server cloud migration. In *Proceedings of WWW'12*. ACM, New York, NY, USA, 979–988.

M. Miglierina, G.P. Gibilisco, D. Ardagna, and E. Di Nitto. 2013. Model based control for multi-cloud applications. In *Proceedings of MiSE'13*. 37–43.

Dorian Minarolli and Bernd Freisleben. 2011a. Utility-based resource allocation for virtual machines in cloud computing. In *Proceedings of ISCC'11*. IEEE, 410–417.

Dorian Minarolli and Bernd Freisleben. 2011b. Utility-based resource allocation for virtual machines in Cloud computing. In *Proceedings of ISCC'11*. IEEE, 410–417.

Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. 2011. Kingfisher: Cost-aware elasticity in the cloud. In *Proceedings of INFOCOM 2011*. IEEE, 206–210.

John D Strunk, Eno Thereska, Christos Faloutsos, and Gregory R Ganger. 2008. Using Utility to Provision Storage Systems.. In *FAST*, Vol. 8. 1–16.

Hong Xu and Baochun Li. 2013. Dynamic cloud pricing for revenue maximization. *IEEE Transactions on Cloud Computing* 1, 2 (2013), 158–171.

Zhen Ye, Athman Bouguettaya, and Xiaofang Zhou. 2014. Economic Model-Driven Cloud Service Composition. *ACM Trans. Internet Technol.* 14, 2-3, Article 20 (Oct. 2014), 19 pages.

Received February 2007; revised March 2009; accepted June 2009