A Mixed Integer Linear Programming Optimization Approach for Multi-Cloud Capacity Allocation

Michele Ciavotta*, Danilo Ardagna, Giovanni Paolo Gibilisco

Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria

Abstract

The large success of the Cloud computing, its strong impact on the ICT world and on everyday life testifies the maturity and effectiveness this paradigm achieved in the last few years. Presently, the Cloud market offers a multitude of heterogeneous solutions; however, despite the undeniable advantages, Cloud computing introduced new issues and challenges. In particular, the heterogeneity of the available Cloud services and their pricing models makes the identification of a configuration that minimizes the operating costs of a Cloud application, guaranteeing at the same time the Quality of Service (QoS), a challenging task. This situation requires new processes and models to design software architectures (SAs) and predict costs and performance considering together the large variability in price models and the intrinsic dynamism and multi-tenancy of the Cloud environments. This work aims at providing a novel mathematical approach to this problem presenting a queueing theory based Mixed Integer Linear Program (MILP) to find a promising multi-cloud configuration for a given software architecture. The effectiveness of the proposed model has been favorably evaluated against first principle heuristics currently adopted by practitioners. Furthermore, the configuration returned by the model has been also used as initial solution for a local-search based optimization engine, which exploits more accurate but time-consuming performance models. This combined approach has been shown to improve the quality of the returned solutions by a 37% on average and reducing the overall search time by 50% with respect to state of the art heuristics based on tiers utilization thresholds.

Keywords: Multi-Cloud Capacity Allocation, Optimization, MILP.

*Corresponding author

Email addresses: michele.ciavotta@polimi.it (Michele Ciavotta), danilo.ardagna@polimi.it (Danilo Ardagna), giovannipaolo.gibilisco@polimi.it (Giovanni Paolo Gibilisco)

1. Introduction

In the last few years Cloud computing has emerged as one of the most innovative computing paradigm. The advent of Cloud has meant several advantages for companies, mainly streamlining and speeding up a part of the process of realization and maintenance of applications, eventually resulting in a large cost reduction. For this reason, every day we witness companies moving their infrastructure, or part of it, on public and hybrid Cloud platforms. Nonetheless, new issues have appeared that are attracting the interest of both researchers and practitioners. One of the main problems that arises when moving to the Cloud is the watchful identification of services to adopt. The ever-increasing number of Cloud providers and the growth of services offered by each of them, in fact, makes complex to spot the most suitable and cost-effective configurations to host the application at hand. Furthermore, moving from an on-premise infrastructure to the Cloud also poses some problems regarding the Quality of Service (QoS) and responsibility in case of downtimes and failures. Cloud providers tend to address these issues negotiating a Service Level Agreements (SLAs) with their customers, pledging to honor certain levels of QoS. However, most of these SLAs provide for discounts on the price of leased services as a penalty for not complying with the agreement. Amazon EC2 SLA, for instance, offers 99.95% of up-time over a month and, claims that the company *will use commercially reasonable efforts* to provide the pledged up-time; however, in case such a value is not achieved users are granted a 10% discount on service cost. This policy might be reasonable for non-critical applications but users with stricter requirements on availability, or other QoS metrics, have no choice but to set up and run their own infrastructure. Moreove, cloud service outages are far to be uncommon. At the time of wirting only during the very last month several issues have been reported ranging from few minutes to several hours [1, 2].

A solution to this problem might come from the variety of the Cloud offers itself; instead of relying on one provider, consider the advantage coming form

exploiting several Cloud platforms at once. For instance, as Amazon and Microsoft advertise at the time of writing a similar SLA with 99.95% of availability, an application deployed on both Clouds can benefit from a combined 99.9999% availability. Furthermore, using multiple providers could allow to dynamically distribute the incoming traffic among the Clouds so as to exploit the hourly differences in pricing with the aim of reducing the operating costs.

Many are the reasons that have so far prevented the fulfillment of the depicted multi-cloud scenario, among them there is the propensity of Cloud providers to provide customized and proprietary technology stacks that make longer and more problematic, in a word uneconomical, the process of implementing, deploying and managing a multi-cloud applications. As a consequence of this strategy, the clients are *de facto* locked in a particular environment. For this reason, presently the choice of the provider and services is one of the first steps to take in the process of designing a software architecture (SA) for a Cloud application. Furthermore, as previously introduced, seeking for a tight-fitting selection of Cloud services to host the application is already challenging for the single Cloud case, when several providers must be considered, the set of possible deployment alternatives grows so dramatically fast to make the problem soon intractable. To make matters worse, while information on architectures and costs are openly available, when it comes to performance things get really complicated. Not only, in fact, to model and evaluate applications is a complex process *per se*, when the dynamism of the Cloud comes into play we enter a pioneering field of research with still no established tools and techniques [3, 4].

In literature have been proposed so far several analytic performance models designed to predict the behavior of software systems at architecture design phase; yet few works provides models that take into consideration peculiarities of the Cloud and there is, to the best of our knowledge, still no attempt to address the problem of multi-cloud architecture design optimization. Further, classical models assumes that the workload, the hardware configuration and its performance to be constant in time [5], whilst Cloud environments are inherently multi-tenant, geographically distributed and virtualized, causing performance

variability over time depending on the congestion level and the competition for resources among the different applications. Further, Cloud platforms often provide tools to dynamically adapt a running application to fluctuations in the workload in order to control certain non-functional requirements (as average response time); such adaptive mechanisms must be also reckoned with at the time of estimating the overall operating costs.

Finally, the problem of representing in a meaningful way an application and the related deployment process on a multi-cloud environment along with the implementation of tools capable to effectively assess the related performance and costs represents only one side of the coin. The other is the problem of exploring the solution space of feasible solutions (*i.e.*, those that meet some user-defined non-functional requirements) seeking the minimum cost deployment over time for the application under study. This problem is described in this work by means of a set of 24 intertwined Capacity Allocation (CA) problems representing a multi-cloud configuration for the reference day; it can be demonstrated to be non-linear, NP-Hard and hence intractable even for simpler cases that do not consider the workload variability over time [6, 5]. Such situation imposes the use of state-of-the-art optimization techniques designed to heuristically explore the solution space selectively addressing only the most promising zones.

Within the framework of MODAClouds[1] EU FP7 project [7], we envisioned a workflow, a stack of meta-models following the Model Driven Engineering (MDE) paradigm, and an integrated platform with the aim of easing the realization of multi-cloud platform-independent applications streamlining pivotal processes of architecture design, service selection, implementation, deployment and runtime management. Specifically, the cornerstones of this approach are the Cloud independence and the multi-cloud enabling technologies. The former trait, is achieved by means of a *develop once, run everywhere* approach based of a middleware layer (*a.k.a.* CPIM library [8]) that abstracts the commonalities of the various Cloud environments hiding their peculiarities. The second result

[1] www.modaclouds.eu

is guaranteed by a wide set of tools for application management; *inter alia*, the multi-cloud data synchronization service (Hegira4Clouds [9]) is worth to be mentioned.

In this paper we introduce one of the components of the MODAClouds ecosystem: SPACE4Cloud, which is responsible for the design-time assessment and optimization of multi-cloud applications. The tool cleverly combines distinct optimization techniques, namely local search algorithms and mathematical programming. In particular, here we detail and validate, highlighting the effectiveness within the overall approach, a Mixed Integer Linear Program (MILP), built to solve in a first approximation the multi-cloud time-dependent capacity allocation problem basing on queuing network models.

The remainder of the paper is organized as follows. In Section 2 the background is briefly introduced. The optimization process is presented in Section 3, whereas Section 4 illustrates the experimental campaign the optimization model underwent and analyzes the outcomes. A detailed State-of-the-art review is reported in Section 5. Conclusions are finally drawn in Section 6.

2. Background: Architecture Modeling and Analyses

In this section we discuss the Model Driven Engineering paradigm we developed within the MODAClouds project and how this can be used to model and optimize multi-Cloud applications with the aim of performing the quality analyses at the basis of our approach. A key element of MDE is the use of Domain-Specific Languages (DSL) [10, 11] to describe the *Model* of the problem at hand. These family of languages offers the flexibility required to address specialized domains by providing a limited set of concepts with well-defined relationships. Just as a language (with its syntax and semantics) allows to express the deepest ideas, so a DSL supports the designer in modeling an application in several respects. In this work we make use of Palladio Component Model (PCM) and Palladio Bench [12] for Quality of Service (QoS) evaluation. PCM is a DSL for the description of component-based architecture and analysis of

non-functional requirements; however, PCM is limited to legacy non-Cloud systems and QoS can be assessed only for the workload peak. On the contrary, Cloud based platforms are dynamic, and time-dependent parameters are essential to correctly assess performance and costs (indeed, the resources allocated vary also during the day if the Cloud elasticity is carried out). In a previous work [13] we provided PCM with new constructs for modeling the diverse nature of the Cloud (varying workload, virtualized resources, services, parameters, *etc.*) allowing the user to fully specify multi-Cloud applications and a cost model to evaluate the execution costs.

The next logical step is to use the model as a base for the assessment of some properties of interest. Depending on which property has to be analyzed, the model can be used as it is or it has to be transformed into a different one, suitable to be automatically evaluated. Several models, specifically designed for the QoS assessment, have been presented in literature; within MODAClouds Layered Queuing Networks (LQNs) [14] have been adopted. LQN formalism allows performance etimation considering both hardware and software contention. Solvers like LINE [4] or LQNS [14] can be used to solve this family of models numerically, without the need of simulating them and derive estimation of performance indicators like response time or resource utilization. The aim of this paper is to present an optimization approach capable of helping application developers to find an optimized deployment configuration that minimize costs and guarantee QoS. The adequacy of LQN models to represent real systems and analyze their performance, expecially int he context of multi-tier and cloud application has been shown in [15].

In our work we exploit the built-in Palladio transformation engine to automatically derive LQN performance models from the higher level PCMs. In the context of web applications usually the workload has daily pattern, for this reason one of the extensions we implemented for the PCM formalism concerns the definition of a variable workload over 24 hours. The LQN model alone cannot be used to represents such variability, we overcome this limitation by generating 24 LQN models, one for each hour of the day. This little trick also enables us to

represent effectively the elasticity of a Cloud by varying the number of virtual resources in each hourly model. The choice of using time slots of one hour has also been guided by the fact that most of the cloud providers offer a pay by the hour policy. Moreover, since the focus of our research is on multi-Cloud applications, for each considered provider a whole set of 24 LQNs is generated and solved.

So far we considered the scenario in which the application designer defines the application, decides the set of providers and for each of them the type and number of virtualized resources to use. The objective is evaluating this configuration to check whether it meets some non-functional requirements expressed in terms of costs and performance. The result can be satisfactory or not; in the latter case the designer can perform a *what-if* analysis, iteratively modifying the original model. This process can be very long and tedious even having at disposal a fast solver due to large number of alternative offers available on the market. A second scenario sees the designer establishing certain conditions in form of constraints, whether they are architectural (predicating, e.g., on components deployment or restricting the candidate set of VMs for application execution) or QoS related (e.g., setting an upper bound on application request average execution time), and lets the tool to explore independently the space of solutions with the goal of fulfilling the requirements and minimizing the cost. To support this case PCM had to be further extended to include new concepts such as *Constraint*, *Target Resource*, *Metric* and *Aggregation* to cite a few. For more details on the proposed PCM extension the reader is referred to [16]. A concrete example of constraints, instead, is discussed at the end of this Section.

In the rest of this section we exemplify the use of our extended PCM models by considering one of the industry case study developed within MODAClouds.

In particular we discuss the case of a software system named *ADOxx* and developed by BOC[2] [17] reviewing and clarifying the modeling concepts expressed so far. ADOxx is a meta-modeling platform used to build and customize BOC

[2]boc-eu.com

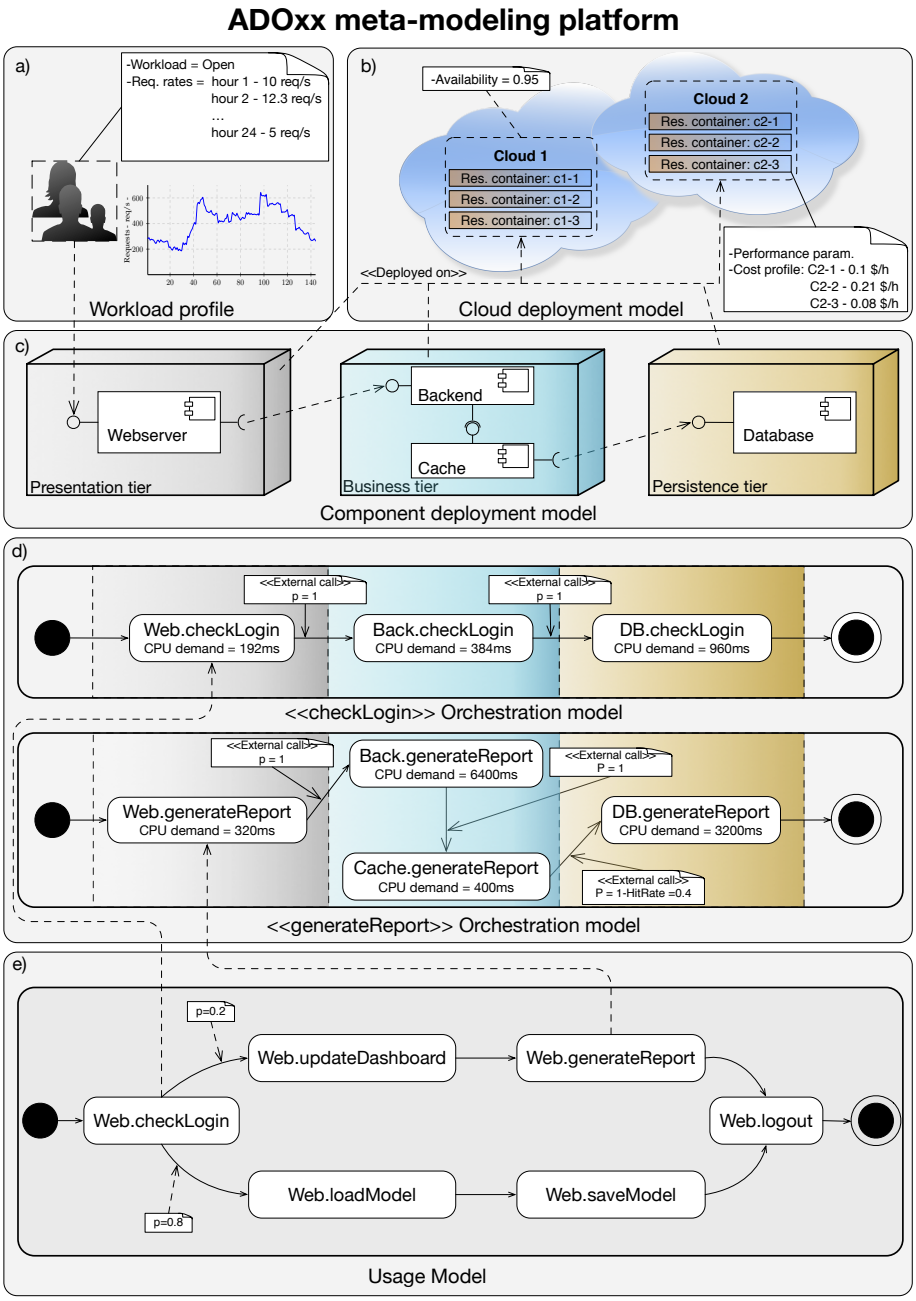


Figure 1: ADOxx meta-modeling platform: Extended PCM instance of the case study.

other tools. Typically, this platform is installed alongside the required components on customers's premises leaving with them the burden of provisioning and maintaining the infrastructure. However, in an effort to attract a larger clientele the company decided to move to the Cloud their platform.

Figure 1 depicts in a compact way the main elements of the software architecture describing ADOxx. Neither all PCM models nor all the associated pieces of information are presented in the Figure; yet this figure is able to represent the core of the application, define the system boundaries and provide insights about the possible interactions between the user and the system as well as some of the interactions internal to the system itself.

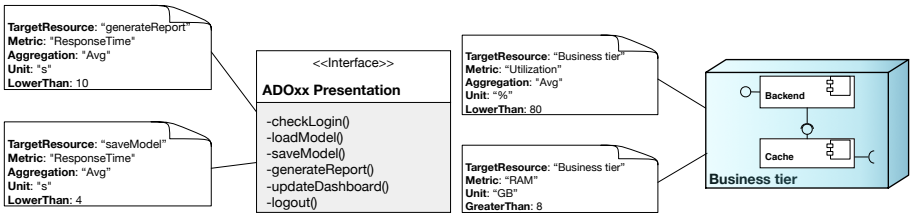
A classical partition in 3-tiers (Presentation, Business and Persistence layers) comprises the base structure of the platform. The *Deployment model* reported in sub-figure 1(c) presents the allocation of the application components on computational nodes modeling cloud-independent resources, highlighting their interactions. The presentation tier, implemented by a webserver running *ADOxx Presentation* component, manages interactions with end users; the business tier hosts all the application logic and a cache system, whilst the persistence tier, hosting the *ADOxx Database* component, manages users' data. Each tier interacts with the others in order to provide a set of functionalities to the end user. In order to further specify the internal behavior of the system, showing the interactions among the interfaces exposed by the tiers, two examples of orchestration models are also reported in the sub-figure 1(d). In particular, these models outline the (stochastic) chains of calls of functionalities necessary to execute a certain user request. Information about the resource demands (related to a reference resource) and call probabilities is also reported for the *checkLogin* and the *reportGeneration* functionality, respectively.

Since the characteristics of load the system is subject to represent a factor of paramount importance for the analysis of performance, this information is also provided (see sub-figure 1(a), *Workload profile*). In particular, the incoming traffic is defined by two properties: the workload type, which is *Open* and a request rate, that varies overtime following a daily *bimodal* distribution.

However, the workload composition (*i.e.*, the percentage of requests for each functionality exposed by the presentation tier) is further characterized within the *Usage model*, which also describe the users' behavior in interacting with the ADOxx platform. In the context of our case study the workload is composed by requests generated by users that exploit ADOxx for their modeling and reporting activities. As shown in sub-figure 1(e) two main classes of users can identified: Modelers, representing the 20% of users, log in to the system, alter the models by loading, modifying, and saving, and eventually, log out. Most of the users, however, belong to the second class. Those users, after the login, interact with the dashboard accessing the information they need, and generating a report before leaving.

Finally, the sub-figure 1(b) sketches a *Cloud deployment* for the considered architecture. Since reliability and responsiveness is an issue for the company, the infrastructure is replicated on two different Clouds to reduce the distance between the data-centers and the user, making the system less susceptible to Internet performance variability. This choice has also been motivated by availability concerns. Since the application is freely available to users, any failure could damage the corporate image leading to potential loss. Information about the availability of the considered Cloud environments is also present in the model. It is important to notice that this model is agnostic with respect to the technology used to host each component and can be re-used across many different platforms. Furthermore, the deployment diagram shows that all the three layers are separately deployed on different Cloud resources, in the Figure referred to as resource containers (*e.g.,* VMs in a IaaS environment, or PaaS workers/containers). Ultimately, each resource is decorated with performance, cost and replication information. As a matter of fact, each tier can be hosted by several replicas of the selected resource container in order to manage traffic variations and guarantee satisfactory QoS.

To conclude the overview of modeling concepts a few words must be spent on those features of the language enabling the user to express desired non-functional aspects. As said PCM models have been extended to support a



(a) QoS constraints defined over an interface (b) Architectural constraints defined over an auto-scaling group

Figure 2: Examples of constraints that can be specified on the software architecture

QoS-aware design-time software optimization. In particular, the language has been provided with new constructs allowing the definition of both QoS and Architectural constraints. As a matter of fact, all the interfaces exposed to the final user, or intern to the system itself, can be annotated with QoS properties, whilst Architectural restrictions can be associated with deployment concepts.

Sub-figure 2(a) shows two examples of QoS constraints that can be defined on the interface exposed by the presentation layer. The topmost constraint expresses the fact that the functionality in charge of generating a report should have an average response time lower than 10 seconds. Similarly, the other bounds the average response time of *saveModel* functionality to 4 seconds. Sub-figure 2(b) presents two Architectural constraints, which can be expressed on the abstract resource used to host the software components of the application. Those constraints predicate the the average CPU utilization and on the minimum amount of RAM necessary to host the components of that tier, respectively.

3. Optimization Process

In this section we showcase the hybrid optimization approach we propose to solve the capacity allocation problem for an application to be hosted on multiple clouds.

As in [18] and [19], in this work we propose a two-step approach to the problem. In the first step a model-to-model transformation is performed to obtain a Mixed Integer Linear Problem (MILP) from a set of models in the

Extended PCM format. The core of this transformation is the mapping between the elements of an SA onto those of a M/G/1 queuing network implementing processor sharing policy. Such operation, described in details in Section 3.2, allows to calculate the average response time for an open workload in a closed form. The resulting MILP model is fed into and solved by a suitable solver.

The objective of this phase is to get an initial solution for the local-search-based optimization algorithm, which represents the second step of our approach; this algorithm iteratively improves such first software architecture by heuristically exploring the space of possible multi-clouds alternative configurations. Each configuration is represented by a set of several hourly LQN models for each Cloud provider, as described in Section 2, which are more expressive and accurate, albeit at the expense of a higher computation time.

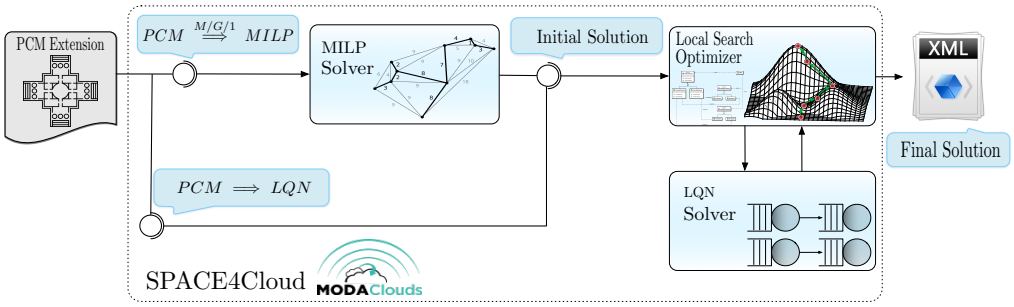


Figure 3: Solution generation workflow

Figure 3 depicts the workflow of the overall optimization process. As explained in Section 2 the specification of the software architecture is provided in PCM format with an accompanying extension. The information contained in these models is used to generate a first-approximation MILP model that is later on solved by a suitable solver. The output of this stage is a multi-cloud configuration consisting in the set of providers, the distribution among them of the incoming workload, and the type and number of resources for each application tier. Such information is used in the creation of the initial solution for the *Local search optimizer*; it is worth to be noticed that, since the building block for every solution handled by the optimized is a LQN representing the elements of the SA, we exploit the PCM2LQN [20] to generate a single LQN that is then

cloned and modified to represent a multi-cloud environment observed over a daily horizon. Finally, the optimizer, which implements a heuristic local-search engine, returns an optimized solution in which the daily execution costs are minimized, while fulfilling the required QoS levels.

In the reminder of this section the capacity allocation problem addressed in this work is described (Section 3.1) while the analytical and local-search-based optimization stages are detailed in Sections 3.2 and 3.3, respectively.

3.1. Search Problem Formulation

As introduced in Section 2, an application can be modeled by means of a software architecture, whose basic elements, the components, implement a set of functionalities, referred to as $\mathcal{F}$. Components are grouped into tiers as a whole deployment unit. Let us denote by $\mathcal{I}$ the set of tiers that support the execution of several components; each tier is allocated on multiple homogeneous resources, *e.g.*, VMs in the IaaS scenario, that evenly share the incoming workload. Being $\mathcal{P}$ the set of available providers, $\mathcal{V}_p$ is the set of available resources types for a particular provider $p \in \mathcal{P}$. Moreover, let $\mathcal{T}$ be the set defined by the N time intervals in which the reference day has been split (*i.e.* 24), each resource type $v \in \mathcal{V}_p$ is characterized by cost and performance information as the leasing price $C_{v,t}$, which might change over the time horizon, the amount of memory M_v, alongside the number and speed of cores possibly associated with the resource.

Each user interacts with the application executing a sequence of requests according to a defined users' usage model; the set of possible requests is referred to as $\mathcal{K} \subseteq \mathcal{F}$. Each request $k \in \mathcal{K}$ is supported by a set of chains of functionality calls $\mathcal{U}_k$. Each chain represents a sequence of calls to functionalities (*i.e.*, an *execution path* [21]) necessary to carry out request k and is denoted by Σ_k. Furthermore in each execution path each functionality is associated with a certain probability value. The orchestration models presented in sub-figure 1 are example of execution paths. Unfortunately, the considered industrial use case is not complex enough to include all possible scenarios, for this reason hereinafter we refer to the example reported in Figure 4. In the picture we consider two classes

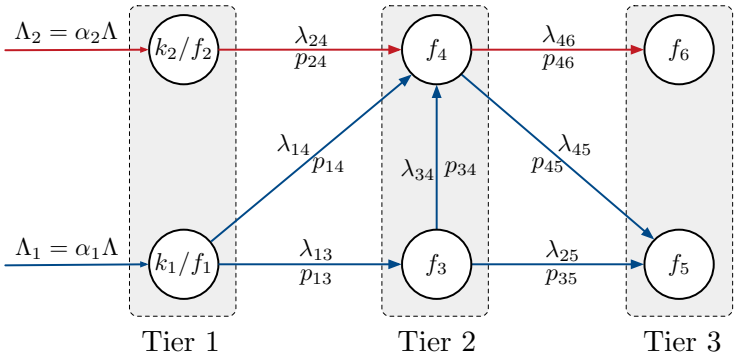


Figure 4: Chains of functionality calls for request classes k_1 and k_2

of requests, namely k_1 and k_2 and six functionalities $\mathcal{F} = \{f_1 \dots f_6\}$ distributed over three tiers. k_1 and k_2 are implemented by (and coincide with) f_1 and f_2 . The blue lines highlight three execution paths for request k_1 whilst the red lines represent the only execution path defined for request k_2. Finally, functionality f_4 lays on three different execution paths from the two request classes.

The number of requests that the application has to process in a particular time interval t is denoted by Λ_t; this workload is unevenly split among the Cloud providers. The amount of workload processed by each provider is denoted by $\Lambda_{p,t}$ with $p \in \mathcal{P}$. In a multi-cloud scenario the minimum number of providers to be selected is given by $\overline{\pi}$, whilst the minimum workload share for each selected cloud provider is given by $\overline{\gamma}$.

We make the common assumption that the request blending of the incoming workload is constant, as in [22, 23, 24], that is the proportions (α_k) among the classes of requests do not vary over time, that is each request k is associated with a workload $\Lambda_{k,t} = \alpha_k\Lambda_t$. In the reference example of Figure 4 the dependence on time and provider has been dropped for sake of readability. Each functionality f is also associated with a share of the incoming workload α_f that does not depend on time; therefore the workload for functionality f can be expressed as: $\Lambda_{f,t} = \alpha_f\Lambda_t$. The latter statement can be easily proven by considering the underlying Discrete Time Markov Chain (DTMC) constituted by all the execution paths

spawned form k. Let p_{kf} be the stationary probability of functionality f over $\mathcal{U}_k$, the workload to be served by f can be calculated as: $\Lambda_{ft} = \sum_{k \in \mathcal{K}} \alpha_k \Lambda_t p_{kf} = \alpha_f \Lambda_t$. In the example functionality f_4 is subject to a workload that is equal to $\lambda_{14} + \lambda_{34} + \lambda_{24}$ that are the components of the workload due to the three afferent execution paths.

Since

$$\begin{cases} p_{k_1 f_4} = p_{14} + p_{13} \cdot p_{34} \\ p_{k_2 f_4} = p_{24} \end{cases}$$

and

$$\begin{cases} \lambda_{24} = \alpha_2 \cdot p_{24} \cdot \Lambda \\ \lambda_{14} + \lambda_{34} = \alpha_1 \cdot (p_{14} + p_{13} \cdot p_{34}) \Lambda \end{cases}$$

α_f can be calculated as: $\alpha_2 \cdot p_{24} + \alpha_1 \cdot (p_{14} + p_{13} \cdot p_{34})$.

We complete the description of the problem, adding the to picture the possibility to further specify the problem with a set of QoS requirements, expressed in terms of thresholds on the average or expected response times $\overline{R}_k$ and maximum unavailability $\overline{U}$, and a set of architectural constraints predicating on the minimum amount of memory required by a cloud resource to host a particular tier, represented by $\overline{M}_i$. Similarly, constraints on the maximum replication factor $\overline{N}_i$ (*e.g.,* the maximum number of cloud resources that can be associated with a certain tier i) can be expressed.

Overall, we outlined a multi-cloud capacity allocation problem whose goal is to find the cheapest deployment capable to fulfill QoS requirements and architectural constraints for each hour of the reference day. To this aim, we identify the following decision variables for the problem:

- x_p, that is a binary variable representing the provider selection sub-problem, it assumes value of 1 if provider p is selected to host the application, 0 otherwise;

System parameters

Index	
$\mathcal{T}$	Reference time horizon
$\mathcal{I}$	Set of application tiers
$\mathcal{F}$	Set of functionalities
$\mathcal{F}_e$	Set of functionalities located in the same tier with functionality e
$\mathcal{K}$	Set of class of requests, $\mathcal{K} \subseteq \mathcal{F}$
Σ_k	Execution path of class of request $k \in \mathcal{K}$
$\mathcal{P}$	Set of cloud providers
$\mathcal{V}_p$	Set of resource types available at provider $p \in \mathcal{P}$
Parameters	
Λ_t	Requests rate at time t
α_f	Percentage of requests of functionality f in the workload
$\mu_{f,v}$	Maximum service rate of requests of functionality f when executed on a resource of type $v \in \mathcal{V}_p$
$C_{v,t}$	Cost of a resource of type v at hosted in provider p at time t
M_v	Memory of a resource of type v in provider p
$\overline{M}_i$	Minimum amount of memory required to host tier i
$\overline{N}_i$	Maximum number of resources (VMs or PaaS containers) used to host a tier i
$\overline{R}_k$	Maximum average response time for requests of class k
$\overline{\gamma}$	Minimum percentage of workload processed by a provider
$\overline{\pi}$	Minimum number of cloud providers
A_p	Availability of provider p
$\overline{U}$	Maximum unavailability specified by the user

Table 1: Optimization model parameters.

Optimization model decision variables.

x_p	Binary variable that is equal to 1 if provider p is selected, 0 otherwise
$z_{i,v,t}$	Integer variable representing the number of virtual resources of type v assigned to the i-th resource pool at time t
$w_{i,v}$	Binary variable that is equal to 1 if the resource type v, of provider p, is assigned to the i-th tier and equal to 0 otherwise
$\Lambda_{p,t}$	Real variable that specifies the amount of workload assigned to the provider p at time t.

Table 2: Optimization model decision variables.

- $w_{i,v}$, that is a binary variable equal to 1 if the resource of type $v \in \mathcal{V}_p$ is assigned to the i-th tier of the application, 0 otherwise;

- $z_{i,v,t}$, an integer variable that specifies the number of replicas of resource of $v \in \mathcal{V}_p$ type (either a IaaS VM or a PaaS worker/container), assigned to the i-th tier at time t;

- $\Lambda_{p,t}$ is an integer variable that specifies the amount of workload assigned to the provider p at time t.

Table 1 and Table 2 summarizes all the introduced parameters and variables.

3.2. Analytic Optimization

The optimization model presented in this section aims at describing a multi-cloud capacity time-dependent allocation problem described in Section 3.1 whose objective is the minimization of the execution costs of a cloud application.

$$\min_{\mathcal{Z}} \sum_{t\in\mathcal{T}} \sum_{p\in\mathcal{P}} \sum_{v\in\mathcal{V}_p} \sum_{i\in\mathcal{I}} C_{v,t} z_{i,v,t} \tag{1}$$

Subject to:

$$\sum_{p\in\mathcal{P}} x_p \geq \overline{\pi} \tag{2}$$

$$\sum_{v\in\mathcal{V}_p} w_{i,v} = x_p \qquad \forall p \in \mathcal{P}, \forall i \in \mathcal{I} \tag{3}$$

$$x_p\, \overline{\gamma}\, \Lambda_t \leq \Lambda_{p,t} \qquad \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \tag{4}$$

$$\Lambda_{p,t} \leq \Lambda_t\, x_p \qquad \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \tag{5}$$

$$\sum_{p\in\mathcal{P}} \Lambda_{p,t} = \Lambda_t \qquad \forall t \in \mathcal{T} \tag{6}$$

$$w_{i,v} \leq z_{i,v,t} \qquad \forall t \in \mathcal{T}, \forall p \in \mathcal{P}, \forall v \in \mathcal{V}_p, \forall i \in \mathcal{I} \tag{7}$$

$$z_{i,v,t} \leq \overline{N}_i\, w_{i,v} \qquad \forall t \in \mathcal{T}, \forall p \in \mathcal{P}, \forall v \in \mathcal{V}_p, \forall i \in \mathcal{I} \tag{8}$$

$$\sum_{v\in\mathcal{V}_p} w_{i,v}\, M_v \geq \overline{M}_i \qquad \forall p \in \mathcal{P}, \forall i \in \mathcal{I} \tag{9}$$

$$\sum_{p\in\mathcal{P}} (ln(1 - A_p)\, x_p) \leq ln(\bar{U}) \tag{10}$$

$$\sum_{v\in\mathcal{V}_p} (1 - \mu_e \overline{R}'_e\, S_v) z_{i,v,t} \leq \Lambda_{p,t}\, \mu_e\, \overline{R}'_e \sum_{c\in\mathcal{K}_e} \frac{\alpha_c}{\mu_c} \qquad \forall t \in \mathcal{T}, \forall k \in \mathcal{K}, \forall e \in \Sigma_k, \forall p \in \mathcal{P} \tag{11}$$

$$x_p \in \{0, 1\} \qquad \forall p \in \mathcal{P} \tag{12}$$

$$w_{i,v} \in \{0, 1\} \qquad \forall p \in \mathcal{P}, \forall v \in \mathcal{V}_p, \forall i \in \mathcal{I} \tag{13}$$

$$z_{i,v,t} \in \mathbb{Z}^+ \qquad \forall t \in \mathcal{T}, \forall p \in \mathcal{P}, \forall v \in \mathcal{V}_p, \forall i \in \mathcal{I} \tag{14}$$

$$\Lambda_{p,t} \in \mathbb{R}^+ \qquad \forall t \in \mathcal{T}, \forall p \in \mathcal{P} \tag{15}$$

This cost represents the objective function (Formula 1) and can be derived as the sum of all costs related to the utilization of Cloud resources considering all application tiers i, time intervals t, selected providers p and the corresponding selection of resource types v. In the most general case the application might be replicated over multiple providers to provide guarantees on its availability. The minimum number of providers to be selected to this aim is bounded by inequality 2. Each Cloud provider offers different type of resources, we use the set $\mathcal{V}_p$ to identify the resources offered by

provider p. The binary variable, $w_{i,v}$ denotes the assignment of a certain type v of resource, among those offered by provider p, to host tier i. The two binary variables just introduced are used in Constraints 3 to guarantee that only a single type of resource is assigned to each application tier for each selected provider. Vice versa these constraints also guarantees that, if a provider is not selected (i.e. $x_p = 0$), its resource types can not host any application tier.

Constraints 4, 5 and 6 are related to the incoming flow of requests and how it is split among the providers over time. In particular, Constraints 4 reflects the minimum partition of workload $\overline{\gamma}$ that has to be served by each provider, and Constraints 5 guarantees that requests are only directed to selected providers. Finally, Constraints 6 forces all incoming requests to be served.

As previously introduced, variables $w_{i,v}$ represent the binding between a resource type and an application tier. This variable alone does not convey the information about the number of replicas of that resource to be used over time. To this end we introduced the set of integer variables $z_{i,v,t}$. The two Constraints 7 and 8 represent upper and lower bounds to the number of replicas of resources of type v, assigned to tier i within each Cloud at time t. Another important effect of these constraints is to force all resources assigned to a certain tier to be of the same type. Constraints 9 provide a bound on the minimum amount of RAM needed by the resources selected to host each tier.

All the constraints presented so far define requirements that shape the structure of the solution but do not address directly the QoS of the application; we call them *Architectural* requirements. The last two families of constraints, instead, are related to the QoS of the application and are called *QoS* requirements.

Constraint 10 is used to ensure a minimum level of availability for the system. Let A_p be the availability of Cloud provider p, the unavailability, defined ad $1 - A_p$, represents the probability of provider p to be unavailable. Since our application can be deployed on multiple hosts we consider the entire application unavailable is all the hosting providers (*i.e.*, those for which $x_p = 1$) are unavailable. Since the failures of different providers can be considered as independent events, the availability of the application is given by $\prod_{p \in \mathcal{P}} (1 - A_p)^{x_p}$. In Constraint 10 this value has been bounded by a maximum unavailability value $\overline{U}$ leading to: $\prod_{p \in \mathcal{P}} (1 - A_p)^{x_p} \leq \overline{U}$. By applying the logarithm to both sides of the formula we get constraint 10.

Inequalities 11, instead, impose an upper bound on the average response time of requests. It has been derived using the M/G/1 queuing model as described next. The processing of a request of class k might involve the execution of other functionalities in the system, deployed on the same as well on different tiers possibly hosted on different resource types; so the average response time of requests belonging a certain class can be express as the summation of the response time of all the functionalities along every execution path Σ_k originated in k, this set is called $\mathcal{U}_k$. The response time of class k is given by:

$$R_k = \sum_{e \in \Sigma_k} \sum_{\Sigma_k \in \mathcal{U}_k} p_{k,e} R_{e,v_e} \tag{16}$$

where $p_{k,e}$ represents (as introduced in Section 3.1) the probability that a request of class k triggers the execution of a functionality $e \in \{\Sigma_k : \Sigma_k \in \mathcal{U}_k\}$,

Notice that for sake of readability we do not consider at this point the dependence neither on provider p nor on time t; moreover we make explicit the dependence on the adopted deployment by means of symbol v_e, which refers to the particular resource hosting the execution of functionality e.

As a consequence of the choice of modeling tiers as M/G/1 queues, we can write a formulation for the average response time of a certain functionality taking into account the functionalities whose components are located on the same tier, that is:

$$R_{e,v} = \frac{\dfrac{1}{\mu_{e,v}}}{1 - \displaystyle\sum_{c \in \mathcal{F}_e} \frac{\Lambda_c}{\mu_{c,v}\, z_{i_e,v}}} \tag{17}$$

where i_e is the tier hosting functionality e whilst $\mu_{e,v}$ represents the maximum service rate of the system when processing a request of functionality e hosted by a virtual resource v. Moreover, notice that Λ_c is the share of workload due to functionality c co-located with e.

In order to reduce the expression we make explicit the dependence of $\mu_{e,v}$ on the hosting resource v using a machine-independent maximum service rate μ_c and a scaling factor S_v that depends on the machine: $\mu_{e,v} = \mu_e S_v$, where S_v represents the proportion between the speed of resource of type v, and a reference resource. Further, we can express Λ_c according to probability α_c and the incoming workload as $\Lambda_c = \alpha_c \Lambda$.

Therefore, Equation 16 can be expressed as:

$$R_k = \sum_{e \in \Sigma_k} p_{k,e} \frac{\frac{1}{\mu_e S_{v_e}}}{1 - \frac{\Lambda}{z_{i_e,v_e} S_{v_e}} \sum_{c \in \mathcal{F}_e} \frac{\alpha_c}{\mu_c}} \leq \overline{R}_k \tag{18}$$

and constrained to be lower than a certain threshold $\overline{R}_k$.

Unfortunately the constraint expressed above is non-linear and, since this characteristic is expected to affect negatively the performance of the model, we opted for bypassing this issue by splitting this constraint into a set of stricter constraints on all the sub-functionalities involved in the execution of request k, that is:

$$R_{e,v_e} \leq \overline{R}'_e \quad \forall k \in \mathcal{K} \quad \forall e \in \Sigma_k \quad \forall \Sigma_k \in \mathcal{U}_k \tag{19}$$

and some algebra we get:

$$\Lambda \, \mu_e \overline{R}'_e \sum_{c \in \mathcal{F}_e} \frac{\alpha_c}{\mu_c} \leq (\mu_e \, \overline{R}'_e \, S_{v_e} - 1) \, z_{i_e,v_e} \tag{20}$$

recalling that $\sum_{v \in \mathcal{V}_p} S_v z_{i_e,v,t} = S_{v_e} \, z_{i_e,v_e,t}$, we obtain Constraints 11.

At this point, we are left with the task of generating the thresholds $\overline{R}'_e$ in such a way that

$$\overline{R}_k \geq \sum_{e \in \Sigma_k} p_{k,e} \overline{R}'_e$$

but also trying to reduce their impact of the feasibility of the MILP solution.

To this end, we opt for splitting the threshold in proportion to the functionality demands in the call chain. We recall that the demand of a certain functionality f, referred to as $D_e = \frac{1}{\mu_e}$, is the average time required to execute e on the reference resource (that in this case a reference VM) [25]. Specifically, we first derive, for every functionality e and for each execution path Σ_k such that $e \in \Sigma_k$, the demand ratio dividing D_e by the demand of the entire execution path D_{Σ_k}, as in Formula 21.

$$r_{e,\Sigma_k} = \frac{D_e}{D_{\Sigma_k}} = \frac{D_e}{\sum_{f \in \Sigma_k} D_f} \tag{21}$$

By using this ratio to split the user-defined response time threshold $\overline{R}_k$ across functionalities in the call chain, we get a set of new constraints on the response time

of each functionality e when executed within the execution path Σ_k: $\overline{R}'_{e,\Sigma_k} = r_{e,\Sigma_k}\overline{R}_k$. As a consequence of this definition we get that

$$\sum_{e\in\Sigma_k} \overline{R}'_{e,\Sigma_k} = \overline{R}'_k. \tag{22}$$

Since a request of class k can have multiple execution paths passing for each functionality involved in its execution chain, we use the most stringent constraint to remove the dependency on the specific execution path, that is:

$$\overline{R}'_e = \min_{\Sigma_k\in\mathcal{U}_k} \overline{R}'_{e,\Sigma_k} \tag{23}$$

From equation 22 and 23 we get:

$$\overline{R}_k \geq \sum_{e\in\Sigma_k} \overline{R}'_e \qquad \forall\Sigma_k \in \mathcal{U}_k \tag{24}$$

Finally, recalling that the an execution chain is essentially a DTMC, we obtain:

$$\overline{R}_k \geq \sum_{\Sigma_k\in\mathcal{U}_k} p_{\Sigma_k} \sum_{e\in\Sigma_k} \overline{R}'_e = \sum_{\Sigma_k\in\mathcal{U}_k} \sum_{e\in\Sigma_k} p_{e,k}\overline{R}'_e \tag{25}$$

where p_{Σ_k} denotes the probability associated with a single execution path.

It is worth to be noticed that the problem presented in this section is $\mathcal{NP}$-hard since, as shown in [26], it is equivalent to a bi-level optimization problem. Nevertheless, with state-of-the-art solvers we are able to find a global optimal solution for this problem in reasonable time, as shown in Section 4.

3.3. Local Search Optimization

The aim of this section is to provide a brief description of the optimization algorithm implemented withing SPACE4Cloud [27, 26], the tool that we used to further optimize the solution obtained from the MILP optimization phase. Even if the global MILP optimal solution can be identified, such solution can be further improved since SPACE4Cloud, as said, exploits a different representation of the original multi-cloud problem. As a matter of fact, it employs LQN models to represent the application and its interactions with the environment. LQNs are able to provide more accurate performance estimates since they explicitly consider applicative layers as well as hardware and software contention. Unfortunately, LQNs are a time-consuming tools and

for this reason the space of possible multi-cloud time-dependent software architectures has to be explored in the most efficient way in order to avoid to evaluate unpromising configurations. For this reason a heuristic approach has been adopted.

The rationale of the optimization engine implemented by SPACE4Cloud consists in characterizing the problem in two-level decision sets and in iteratively using two different local search strategies (one for each set) to improve an initial solution (*Hybridization*). The upper-level local search is devoted to the selection of the most suitable resource type per tier and per provider considering eventual user-defined architectural constraints; a multi-start Tabu-search [28] like sub-routine is in charge of this task. In a nutshell, this procedure randomly selects one of the tiers of the architecture and modifies the related resource type according to a selection process based on a roulette-wheel, or fitness proportionate, mechanism. This selection method is commonly used in genetic algorithms but has been demonstrated to be beneficial also in other approaches. When a local optimum is found (after some iterations of both local searches) the optimization process is re-executed from a new configuration expressly generated to address the search toward poorly explored zones of the solution space. Such a multi-start tabu mechanism is implemented via two memory structures: a *short term* and a *long term* one. The goal of the short memory is to avoid cycles in the upper-level search phase of the algorithm. The long term memory, in turn, is used to store the frequency of assignments and evaluations for a particular provider, resource type and tier with the aim to implement an aspiration criterion (for the multi-start mechanism) that allows the algorithm to escape from local optima, breaking free from constraints imposed by the short term memory.

The lower-level procedure, instead, implements a Greedy Randomized Adaptive Search (GRASP) [29] technique to optimize the number of replicas of the assigned resource for each tier; the goal is to find the minimum number of resources and the best distribution of the workload among the Cloud providers for each tier to fulfill the QoS requirements. This procedure is applied independently, and in parallel, on all providers and all periods of the reference time horizon and terminates when a further reduction in the number of replicas or change in the workload distribution would leads to an unfeasible solution. Notice that since higher-level decisions can lead to the generation of an infeasible solution, before the GRASP phase can be applied the feasibility must be re-established by enacting a progressive increment of the number

resources of each tier.

4. Experimental Results

This section reports the results of the scalability and quality optimization analyses performed for a variety of software architectures with the aim to prove the soundness and usefulness of our approach. Section 4.1 briefly presents the experimental setup. The scalability analysis for the solution of the MILP problem formulation is discussed in Section 4.2, whereas Section 4.3 summarizes the performance of our hybrid optimization in a comparison with first principle policies that can be implemented at IaaS providers based on thresholds on tier utilization.

4.1. Experimental Setup

As introduced in Section 3.1, the problem of finding the optimal allocation of services to application tiers presents several decision dimensions. Using the experience we gathered during the analysis of the case study we identified the main factors influencing the time needed to derive a quality solution using the proposed hybrid approach.

Our optimization problem can be roughly characterized in terms *number of providers* ($|\mathcal{P}|$), *number of tiers* ($|\mathcal{I}|$) of the application under analysis and *number of functionalities* ($|\mathcal{F}|$). However, whereas, the size of $\mathcal{P}$ and $\mathcal{I}$ do have a direct effect on the size of the solution space of the problem discussed in Section 3.1, the $|\mathcal{F}|$ does not affect directly the optimization procedure; yet it has an impact on the complexity of the LQN performance model. Solving an LQN model is a time-consuming task and the optimization procedure is often required to evaluate a great number of them; in the worst case one model per hour of the day and for each Cloud provider must be evaluated and this must be repeated for each of the hundreds of solutions that might be generated throughout the optimization process.

The analysis we performed are intended to be representative of real Cloud applications. To assess the soundness and scalability of the proposed approach we built a benchmark consisting of a set of 42 randomly generated instances obtained varying the performance parameters according to the ranges used by other literature approaches [30, 31, 32, 33] and from a real system [34] (see Table 3). Resource costs and capacities have been taken from Amazon EC2, Microsoft Azure, and Flexiscale.

Since most of real world applications are composed by two or three tiers [35] [34] we restrict ourself to generating instances with only two and three tiers varying the number of functionalities from a minimum of 6 up to a maximum of 12. All the generated software architectures (*i.e.*, the benchmark instances) expose three functionalities allocated on the first tier that typifies a web server or an application proxy. The invocation of each functionality by the end user triggers the execution of a chain of functionalities hosted on the other tiers. Finally, the assignment of functionality to tiers depends on the considered $|\mathcal{F}|$ and $|\mathcal{I}|$ in such a way to balance the load generated across all the tiers of the system. This choice has been made in order to challenge the optimization algorithm; in this way the local-search algorithm is forced to optimize all the application tiers at once since none of them appear to be more critical than the others.

Since QoS constraints are important to avoid flat naive configurations, we paid attention in generating them meaningfully. Suitable constraints on the execution time of the three functionalities offered by the system are therefore derived by summing up the demands along the execution paths of each functionality, across all the involved tiers, and multiplying this value by 10, as in [21, 30]. Amazon EC2 m3.medium has been used as a reference virtual machine to generate resource-independent demands. We have, thereafter, introduced an architectural constraint specifying that the first and third tier of each architecture have to be hosted on virtual machines with at least 2GB of memory. We did not specify any constrain on the second tier in order to allow the algorithm to explore a wider space of configurations. Both single and multi-cloud scenarios (with 2 and 3 providers) are considered. In the two multi-cloud scenario, we imposed an additional constraint requiring that, if a provider is selected, it has to serve at least 20% of the incoming workload.

Workloads have been generated by considering the trace of a large Web system including almost 100 servers. The trace contains the number of sessions, on a per-hour basis, over a one-year period. The trace follows a bimodal distribution with peaks around 11.00 a.m. and 4.00 p.m. Multiple workloads have been obtained by adding random white noise to each sample as in [30] and [36].

The ranges of the considered model parameters are reported in Tables 3 and 4 for sake of completeness.

In order to guarantee statistical independence of our scalability results (note that

Parameter	Range
α_k	$[0.1; 1]$ %
$p_{k,f}$	$[0.01; 0.5]$
$\mu_{f,v}$	$[50; 2800]$ req/sec
$C_{v,t}$	$[0.06; 1.06]$ $ per hour
$\overline{M}_i$	[1;4] GB
N	5000
$\overline{R}_k$	$[0.005; 0.01]$ sec

Table 3: Ranges of model parameters.

Parameter	Range
Number of providers $\|\mathcal{P}\|$	$[1; 3]$
Number of tiers $\|\mathcal{I}\|$	$[1; 3]$
Number of time Intervals $\|\mathcal{T}\|$	$[1; 24]$
Number of Requests Classes $\|\mathcal{K}\|$	3
Number of Functionalities $\|\mathcal{F}\|$	$[6; 12]$
Number of resource types $\|\mathcal{V}_p\|$	$[1; 12]$

Table 4: MILP sets cardinalities.

the second optimization step includes random moves), for each test each of the 42 different instances we considered was solved 25 times, leading to a total of 1.050 experiments.

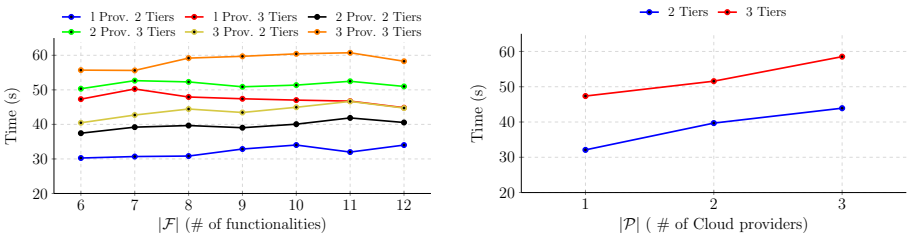
All the experiments reported in this section have been performed on a VirtualBox virtual machine based on Ubuntu 12.10 server with four virtual CPUs hosted on a Xeon E5530 and 6GB of memory. ILOG CPLEX 12.2.0.0 [3] has been used as MILP solver.

4.2. Scalability analysis

Figure 5 shows the detailed results of the scalability analysis for architectures of 2 and 3 tiers and deployments on 1 to 3 Cloud providers. Each value reported in the figures is averaged over 25 runs.

In particular, Figure 5(a) shows that the time spent in solving the MILP formulation is only marginally affected by the number of functionalities the architecture implements. This behavior was expected, since the number of functionalities has the only effect to increase the number of response time constraints, modeled by equations 11 in the model presented in Section 3.2. Since this increment is quite limited, the effect on the solver execution time is modest. Both $|\mathcal{I}|$ and $|\mathcal{P}|$ have, instead, an important impact on the solution time with larger times observed for the 3-tier instances (see Figure 5(b)). However, it is worth to be noticed that the time increases almost linearly even in the worst case the time is always restrained, legitimizing the use within a wider optimization approach. The simplest model, in fact, has been solved in 30.26 seconds (for the 2-tier/1-provider problem) whilst the most difficult one in 60.74 seconds (for the 3-tier/3-provider problem).

[3] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/



(a) (b)

Figure 5: Scalability analysis of the proposed MILP formulation.

Detailed results of the scalability analysis are reported in columns named **Time(s)** and $\boldsymbol{\sigma_{Time}}$**(s)** of Table 5. For each instance the table details the number of Cloud providers, tiers and functionalities of the application specified in the model. The average and the standard deviation of the time required to solve the MILP problem are reported.

4.3. Solution quality evaluation

The aim of this Section is to assess the quality of the solution obtained using the proposed MILP formulation and its impact on the outcomes of SPACE4Cloud. Moreover, we evaluate the behavior of the proposed formulation comparing it with first principle heuristics, widely used in practice [32, 33], which can be roughly described by the following two rules of thumb:

1. For every tier, select the cheapest VM type available at the Cloud provider satisfying (if stated) memory constraints represented by Equation 9;
2. Similarly to the auto-scaling policy commonly implemented by IaaS providers [37], the number of VMs allocated for each tier is determined such that the average CPU utilization is below a given threshold $\overline{\rho}$.

To analyze a wider range of behaviors we implemented two possible contending heuristics as in [33], namely Heur_{60} and Heur_{80} by setting $\overline{\rho} = 0.6$ and $\overline{\rho} = 0.8$, respectively.

Figure 6 shows the trace of the execution of the optimization implemented by SPACE4Cloud using as initial point the solution obtained by the Heur_{60} heuristic, in gray, and the one derived by our approach, in black. On the x axis the overall optimization time is reported (including the time required to determine the initial solution by the MILP or the heuristic and the local search execution time), while the y axis reports the Cloud daily resource cost.

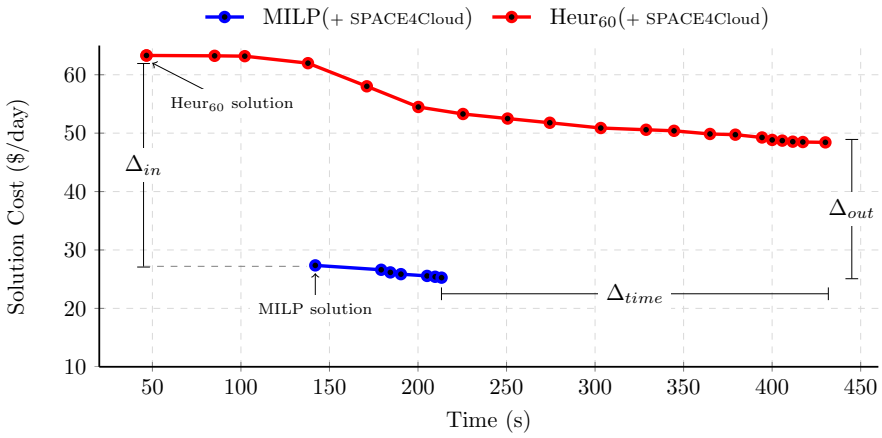


Figure 6: Comparison of execution traces of SPACE4Cloud when using the MILP initial solution or the Heur_{60} initial solution

This trace has been obtained by instrumenting SPACE4Cloud in order to report how the cost of the solution changes during the optimization process. The trace in Figure 6 comes from one of the comparison runs we performed but is representative of most of the behaviors. More details on the results of other runs are given in Table 5. In particular we have first analyzed the difference in cost between the first feasible solution found by the optimization run starting from the heuristic solution and by the MILP approach, marked as Δ_{in}. This value shows the distance between the staring points of the two optimization runs. Positive Δ_{in} values indicate that the solution found by the MILP approach is cheaper than the one obtained by the heuristic. The initial gap on the x axis represents the time required to generate the initial solution by solving the MILP formulation of the problem. The Δ_{out}, shows the difference of the cost of the best solution found by the two optimization runs, this value represents the final savings obtained by using the MILP initial solution in place of the Heur_{60}. Positive values indicate reduction in cost obtained by the MILP heuristic. Finally we analyze the Δ_{time} which represents the time saved in the optimization process. Positive values indicate a reduction in the time required by the optimization process performed by SPACE4Clouds when using as initial solution the one derived by the MILP formulation of the problem. Negative values indicate a growth in the time required for the optimization.

Even if the evaluation of the MILP solution introduces an initial delay, the per-

formance of the local search is significantly improved. In this specific case the final solution is around 55% cheaper, while the time required by the local search to identify the final local optimum is reduced by half.

Table 5 summarizes the results achieved in the columns on the right side, under the *Quality Evaluation* header. The results reported in each column have been obtained by considering 210 total runs of SPACE4Cloud starting from an initial solution generated either by using our MILP approach, the Heur_{60} or the Heur_{80} heuristics. Columns Δ_{in}, Δ_{out} and Δ_{time} report the relative percentage gain/loss when using the MILP approach to generate the initial solution with respect to the heuristic approach.

With respect to the Heur_{60} heuristic, the average cost reduction of the final solution obtained by SPACE4Cloud when using as initial solution the one generated by our MILP approach is of 37%. The average reduction in the time required to find the solution is of 50% With respect to the Heur_{80} heuristic, the average cost reduction of the final solution obtained by SPACE4Cloud is of 37% and the average reduction in the time required to find the solution is of 22%. When three Cloud providers are considered, the time required to perform the optimization of SPACE4Cloud when using the initial solution generated by MILP is sometimes larger than the one required when starting from the Heur_{80}. Nevertheless, comparing the final solution obtained by those runs show that the solution found using the MILP approach is, on average, 27% cheaper.

5. Related Work

Our work lays in the Model-Driven Quality Prediction (MDQP) research area. MDQP starts from a description of the software system in terms of UML models, in order to support quality prediction; the Object Management Group (OMG) introduced two particular flavors of UML tailored for this purpose: the UML Schedulability, Performance and Time (SPT) profile [38] and the UML Modelling and Analysis of Real-time Embedded (MARTE) systems profile [39]. These profiles support the modeling of resource allocation and the definition of non-functional properties.

Many other approaches share the same idea of extending design-time models in order to support performance analysis and QoS requirements; in particular the Palladio Component Model (PCM) [12], developed by Becker et al., provides a language that can be used to model the architecture of an application, its deployment and users'

			Scalability		Quality Evaluation					
					MILP *vs* Heur_{60}			MILP *vs* Heur_{80}		
$\|\mathcal{P}\|$	$\|\mathcal{I}\|$	$\|\mathcal{F}\|$	**Time(s)**	σ_{Time}(s)	Δ_{in} %	Δ_{out} %	Δ_{time} %	Δ_{in} %	Δ_{out} %	Δ_{time} %
1	2	6	30.26	0.89	76.9%	32.5%	57.8%	69.6%	35.2%	25.9%
		7	30.69	0.88	82.5%	49.8%	47.1%	77.0%	45.2%	9.4%
		8	30.83	0.72	83.8%	47.8%	22.7%	78.7%	28.1%	68.3%
		9	32.87	1.05	90.5%	36.0%	67.5%	82.4%	33.7%	66.6%
		10	34.03	1.06	87.1%	34.0%	60.5%	82.9%	29.8%	52.3%
		11	31.99	1.44	89.6%	42.8%	57.1%	86.2%	39.6%	49.9%
		12	34.02	1.69	89.8%	44.0%	46.6%	86.4%	33.3%	56.2%
	3	6	47.31	1.68	72.1%	37.3%	57.7%	63.9%	38.1%	43.1%
		7	50.25	3.58	75.4%	39.3%	49.0%	68.4%	43.7%	18.3%
		8	47.93	2.53	80.6%	42.9%	58.4%	74.5%	39.6%	54.6%
		9	47.42	2.42	83.3%	36.0%	74.1%	78.0%	36.3%	52.0%
		10	47.03	1.64	85.4%	45.4%	21.3%	80.8%	39.5%	43.8%
		11	46.74	1.66	87.2%	44.2%	50.0%	83.1%	42.3%	41.9%
		12	44.83	1.36	89.2%	46.9%	46.2%	85.7%	47.6%	19.0%
2	2	6	37.45	0.71	86.0%	39.1%	68.7%	85.1%	44.5%	43.6%
		7	39.20	0.81	89.0%	36.2%	68.1%	84.4%	43.9%	43.0%
		8	39.67	1.13	91.4%	36.3%	71.5%	87.3%	43.3%	62.7%
		9	39.04	1.36	92.5%	42.4%	76.7%	90.9%	53.1%	67.5%
		10	40.06	0.90	92.8%	40.4%	73.9%	90.6%	48.4%	69.2%
		11	41.87	1.62	93.5%	44.5%	74.9%	92.3%	53.3%	56.2%
		12	40.55	2.37	93.9%	40.8%	78.4%	92.3%	46.5%	60.9%
	3	6	50.32	2.89	81.3%	35.7%	68.5%	80.3%	40.1%	52.6%
		7	52.66	1.34	86.2%	38.7%	59.2%	80.9%	43.1%	22.1%
		8	52.30	1.00	89.7%	38.2%	64.0%	86.2%	45.9%	43.3%
		9	50.89	1.42	90.1%	41.1%	65.0%	88.1%	46.8%	48.6%
		10	51.37	1.26	91.2%	40.5%	67.3%	86.9%	45.3%	57.9%
		11	52.48	1.07	92.5%	44.5%	77.9%	88.7%	47.7%	67.3%
		12	50.99	1.00	93.0%	40.2%	80.4%	91.8%	48.0%	68.1%
3	2	6	40.47	8.33	76.6%	22.6%	37.7%	73.2%	17.9%	-27.2%
		7	42.71	0.84	81.0%	19.6%	25.3%	74.6%	17.3%	-59.5%
		8	44.45	3.21	84.3%	30.3%	31.5%	80.2%	28.2%	-33.4%
		9	43.45	1.13	87.8%	37.8%	26.0%	86.8%	35.6%	-20.4%
		10	44.96	0.95	89.0%	37.5%	38.9%	83.4%	34.7%	-13.7%
		11	46.65	2.13	90.5%	33.3%	-2.5%	88.3%	31.8%	-79.1%
		12	44.73	2.99	91.8%	35.1%	41.3%	88.0%	33.6%	-40.1%
	3	6	55.71	1.69	74.2%	25.8%	39.2%	66.5%	22.5%	-68.2%
		7	55.62	1.86	76.9%	28.2%	25.5%	73.7%	25.3%	-71.9%
		8	59.19	1.90	82.0%	31.2%	40.8%	79.2%	28.8%	36.9%
		9	59.71	1.67	83.0%	28.7%	33.6%	80.8%	25.8%	-2.0%
		10	60.42	1.91	84.6%	29.8%	40.4%	80.4%	27.7%	-7.3%
		11	60.74	1.42	86.9%	23.9%	26.0%	82.7%	21.6%	-28.7%
		12	58.28	1.83	88.3%	30.2%	26.9%	87.8%	27.5%	-34.9%

Table 5: Results of the Scalability and Quality Evaluation analysis.

behavior with particular concern for QoS characteristics. Such models can then be analyzed by means of a simulation engine [40] or transformed into a Layered Queuing Network (LQN) model and analyzed by LQN solvers (like LINE [4] or LQNS [41]) to derive performance metrics. Other approaches that perform transformation from design-time models to performance ones can be found in surveys [42, 43, 44] that provide a good coverage of the most important approach for performance prediction at design-time. The outcome of the MDQP process has been used as starting point for design-time performance optimization in different works.

In order to present how similar optimization problems have been approached in the literature, we use a classification partially derived by the one presented in [45], which is suited to describe a broad range of similar approaches. This taxonomy has the ambition to be as general as possible presenting: semi-automated framework embedding rules and the knowledge of anti-patterns (Rule-based category), problem-independent optimization techniques (Metaheuristic category) and solutions relying on reasoning techniques to tackle the problem of constraint satisfaction (Generic Design Space Exploration category).

Rule-based approaches. This category groups together approaches that embed performance knowledge into feedback rules. The general flow of optimization approaches that falls in this category is to evaluate a candidate solution to derive performance indicators and then apply rules, by means of model-to-model transformations, in order to improve the system architecture in a semi-automatic way. The baseline for this approach is the Query, View, Transformation language defined by OMG that has been adopted and extended by many approaches such as [45] with the addition of feedback rules. Among the rule based approaches we can identify the framework PUMA by Woodside et al., [46] that support JESS feedback rules. Other approaches, like [47] or [48], focus on the identification of anti-patterns on existing systems, specified by a set of rules, the main limitation of these approaches is that most of them they are language specific. A key difference between the works related to the identification of performance anti-patterns and our work is that usually performance anti-patterns are mostly related to the architecture of the application and do not consider its deployment, our approach on the other hand is focused on the identification of an optimized deployment of the application. In our approach, however, architectural anti-pattern can be identified as side effect. In the context of distributed systems a rule based ap-

proach for configuration optimization, according to QoS metrics, has been proposed in [49]. In[50] is proposed an approach that comprises a trade-off analysis of competing Non-Functional Requirements (NFR) in order to find critical components for one or more NFR. Their work involves manual intervention for the specification and selection of transformation rules with the objective to find an architecture that satisfies NFR in a process of consecutive refinements.

Metaheuristics. Metaheuristic approaches aim at exploring the space of possible solutions using high-level algorithms, often inspired by biology or physics. The Automated Quality-driven Optimization of Software Architecture (AQOSA) framework [51], for instance, allows the optimization of multiple criteria exploring the design space by means of an evolutionary algorithm in order to derive a set of Pareto optimal solutions. A similar approach, based on a genetic algorithm is the ArcheOpterix presented in [52]. A specialization of such work in the context of embedded system has been proposed in [53] in order to optimize reliability and energy consumption. Another genetic algorithm based approach with focus on service composition id presented in [54]. Genetic algorithms usually effective in solving multidimensional optimization problems but need to evaluate a high number of solution. If the time needed to evaluate objective functions over the solutions is considerable evolutionary approaches are not applicable. In such a situation approaches that keeps alive a single or just a couple of solutions, like the one by Ouzineb et al. [55] based on a Tabu Search heuristic, are more convenient. An approach similar to the one presented in this work has been proposed in [19], where an hybrid bi-level Tabu Search was used to optimize the deployment of an application on a single cloud provider.

[18] address the problem of deriving deployment decisions using an approach similar to the one presented in this work. Authors make use of an analytic optimization problem to derive a promising initial population for an evolutionary algorithm. In this work they deal with the optimization of conflicting three objectives, namely cost and response time and availability. The main differences between this and our work lies in the fact that in [18] only legacy in-house enterprise systems are considered while here we take into account the uniqueness of the Cloud environment to develop a fully new software architecture optimization approach able to deal also with the multi-cloud scenario. Ultimately, in [6] is proposed a combined metaheuristic-simulation approach to solve the problem of migrating existing enterprise software to Cloud platforms. A

combination of a specific Cloud environment, deployment architecture, and runtime reconfiguration rules are considered. The design space is explored by means of a genetic algorithm while a simulator is charged with the solution performance evaluation. Although there are some similarities, many are the differences. To start with our approach is not suitable for legacy systems since it has been designed to help QoS designers to design multi-Cloud ready application, moreover we explicitly consider both architectural and QoS constraints during the search process. Finally our approach takes into consideration deployment scenarios over a daily horizon leading to multiple capacity allocation solutions, each one tailored to one hour of the day.

Generic Design Space Exploration (GDSE) are frameworks that are not particularly tailored for some problem instances but provide a way to explore a general space of possible solutions encoding feedback rules into a Constraint Satisfaction Problem. An example of such a framework is DeepCompass [56] that is suited to optimize component based application on multiprocessor systems. A similar approach based on boolean trees is presented in [57], the approach is general but can be specialized to take QoS aspects into consideration. Also a general approach is presented in [58], that provides a language to specify constraints and allows the generation of candidate solution by means of different solvers.

A different approach is Formula, presented in [59]. It consists in the specification of the problem as a satisfiability problem and use the Z3 Satisfiability Modulo Theory solver to derive solutions compliant with the design specification. To do so, Formula makes use of logic programs to specify non-functional requirements and transform these constraints along with the application models and meta-models into first-order logic relations.

6. Conclusion

In this paper we present a matheuristic approach for the multi-cloud capacity allocation problem wherein a MILP formulation, based on queue-theory results, is solved with the goal of identifying a promising initial solution to be, thereafter, fed to a local-search optimization procedure. The proposed hybrid approach is meant to yield a reduction of development time, the running costs, and the overall quality of

a multi-cloud application by providing an automated and effective search procedure, able to identify more and better design alternatives.

To demonstrate the suitability of the proposed MILP model for large-sized problems, a scalability analysis has been performed and discussed, showing that, as a matter of fact, existing state-of-the-art solvers can solve the largest formulation in at most one minute.

Furthermore, the hybrid approach to optimization presented in this work has been proven to be effective in improving the local-search outcomes in terms of both quality and optimization time (we observed a 37% reduction of the costs on average) with respect to commonly adopted first principles heuristics based on utilization thresholds.

Future work will concern on the one hand the validation of the proposed approach on additional industry case-studies whilst on the other it will be extended to entail the QoS assessment and optimization of Big Data software systems.

Acknowledgments

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8- 318484 (MODAClouds).

References

[1] Microsoft, Microsoft Azure Status,`https://azure.microsoft.com/en-us/status` , [Online; accessed 29-December-2015] (12 2015).

[2] Google, Google Cloud status, `https://status.cloud.google.com/summary` , [Online; accessed 29-December-2015] (12 2015).

[3] G. Casale, M. Tribastone, Fluid analysis of queueing in two-stage random environments, in: Proceedings of QEST 2011, 2011.

[4] J. Perez, G. Casale, Assessing SLA Compliance from Palladio Component Models, in: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on, 2013, pp. 409–416.

[5] A. Martens, H. Koziolek, S. Becker, R. Reussner, Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, in: WOSP/SIPEW 2010, 2010.

[6] S. Frey, F. Fittkau, W. Hasselbring, Search-based genetic optimization for deployment and reconfiguration of software in the cloud, in: ICSE 2013, 2013.

[7] D. Ardagna, E. di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, C. Sheridan, Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds, in: Modeling in Software Engineering (MISE), 2012 ICSE Workshop on, 2012, pp. 50–56.

[8] F. Giove, D. Longoni, M. S. Yancheshmeh, D. Ardagna, E. Di Nitto, An approach for the development of portable applications on paas clouds., in: CLOSER, 2013, pp. 591–601.

[9] M. Scavuzzo, E. D. Nitto, S. Ceri, Interoperable Data Migration between NoSQL Columnar Databases, in: EDOC Workshops 2014, Ulm, Germany, September 1-2, 2014, 2014, pp. 154–162.

[10] F. Fleurey, A. Solberg, A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems, in: Model Driven Engineering Languages and Systems, Springer, 2009, pp. 606–621.

[11] M. Acher, P. Collet, P. Lahire, R. B. France, Familiar: A domain-specific language for large scale management of feature models, Science of Computer Programming 78 (6) (2013) 657–681.

[12] S. Becker, H. Koziolek, R. Reussner, The Palladio component model for model-driven performance prediction, Journal of Systems and Software 82 (1) (2009) 3–22.

[13] D. Franceschelli, D. Ardagna, M. Ciavotta, E. Di Nitto, Space4cloud: a tool for system performance and costevaluation of cloud systems, in: Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds, MultiCloud '13, ACM, New York, NY, USA, 2013, pp. 27–34.

[14] G. Franks, T. Al-Omari, M. Woodside, O. Das, S. Derisavi, Enhanced modeling and solution of layered queueing networks, Software Engineering, IEEE Transactions on 35 (2) (2009) 148–161.

[15] J. F. Pérez, G. Casale, S. Pacheco-Sanchez, Estimating computational requirements in multi-threaded applications, IEEE Trans. Software Eng. 41 (3) (2015) 264–278.

[16] D. Ardagna, M. Ciavotta, M. Miglierina, G. Gibilisco, G. Casale, J. Pérez, F. D'Andria, R. S. González, MODAClouds D5.2.2 - MODACloudML QoS abstractions and prediction models specification - final version, Available at `www.modaclouds.eu` (2014).

[17] A. Gunka, S. Seycek, H. Kühn, Moving an application to the cloud: An evolutionary approach, in: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds, MultiCloud '13, ACM, New York, NY, USA, 2013, pp. 35–42.

[18] A. Koziolek, D. Ardagna, R. Mirandola, Hybrid multi-attribute qos optimization in component based software systems, Journal of Systems and Software 86 (10) (2013) 2542 – 2558.

[19] D. Ardagna, G. P. Gibilisco, M. Ciavotta, A. Lavrentev, A multi-model optimization framework for the model driven design of cloud applications., in: SSBSE, 2014, pp. 61–76.

[20] H. Koziolek, R. Reussner, A model transformation from the palladio component model to layered queueing networks, in: S. Kounev, I. Gorton, K. Sachs (Eds.), Performance Evaluation: Metrics, Models and Benchmarks, Vol. 5119 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 58–78.

[21] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, Software Engineering, IEEE Transactions on 33 (6) (2007) 369–384.

[22] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, IEEE Trans. Software Eng. 33 (6) (2007) 369–384.

[23] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Wiley-Interscience, New York, NY, USA, 1998.

[24] D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malek, J. P. Sousa, A framework for utility-based service oriented design in SASSY, in: Proceedings of the first

joint WOSP/SIPEW Inter. Conference on Performance Engineering, 2010, ACM, 2010, pp. 27–36.

[25] E. Lazowska, J. Zahorjan, G. Graham, K. Sevcik, Quantitative system performance: computer system analysis using queueing network models, Prentice-Hall, Inc., 1984.

[26] G. P. Gibilisco, A methodology and a tool for qos-oriented design of multi-cloud applications, available at `http://home.deib.polimi.it/ardagna/GibiliscoPhDThesis`, Ph.D. thesis, Politecnico di Milano.

[27] D. Ardagna, M. Ciavotta, G. P. Gibilisco, G. Casale, J. F. Pérez, MODAClouds D5.4.3 - Prediction and cost assessment tool - final version, Available at `www.modaclouds.eu` (2015).

[28] F. Glover, Tabu search: part i, ORSA Journal on computing 1 (3) (1989) 190–206.

[29] M. Resende, C. Ribeiro, Greedy randomized adaptive search procedures, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Vol. 57 of International Series in Operations Research & Management Science, Springer US, 2003, pp. 219–249.

[30] D. Ardagna, S. Casolari, M. Colajanni, B. Panicucci, Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems, Journal of Parallel and Distributed Computing 72 (6) (2012) 796 – 808.

[31] J. Almeida, V. Almeida, D. Ardagna, I. Cunha, C. Francalanci, M. Trubian, Joint admission control and resource allocation in virtualized servers, Journal of Parallel and Distributed Computing 70 (4) (2010) 344 – 362.

[32] A. Wolke, G. Meixner, Twospot: A cloud platform for scaling out web applications dynamically, in: ServiceWave, 2010.

[33] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D.Gmach, R. Gardner, T. Christian, L. Cherkasova:, 1000 islands: An integrated approach to resource management for virtualized data centers, Journal of Cluster Computing 12 (1) (2009) 45–57.

[34] D. Ardagna, B. Panicucci, M. Trubian, L. Zhang, Energy-aware autonomic resource allocation in multitier virtualized environments, Services Computing, IEEE Transactions on 5 (1) (2012) 2–19.

[35] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, L. Zhang, A hierarchical approach for the resource management of very large cloud platforms, IEEE Trans. Dependable Sec. Comput. 10 (5) (2013) 253–272.

[36] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster Computing 12 (1) (2009) 1–15.

[37] Amazon, AWS Elastic Beanstalk, `http://aws.amazon.com/elasticbeanstalk/`, [Online; accessed 29-December-2015] (12 2015).

[38] OMG, UML Profile for Schedulability, Performance, and Time Specification (2005).

[39] OMG, A uml profile for marte: Modeling and analysis of real-time embedded systems (2008).

[40] C. Vogel, H. Koziolek, T. Goldschmidt, E. Burger, Rapid Performance Modeling by Transforming Use Case Maps to Palladio Component Models, in: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, ICPE '13, ACM, New York, NY, USA, 2013, pp. 101–112.

[41] O. Das, C. M. Woodside, Dependable lqns: A performability modeling tool for layered systems, in: 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE Computer Society, 2003, pp. 672–672.

[42] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, Model-based performance prediction in software development: A survey, Soft. Eng., IEEE Trans. 30 (5) (2004) 295–310.

[43] H. Koziolek, Performance evaluation of component-based software systems: A survey, Performance Evaluation 67 (8) (2010) 634–658.

[44] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya, Software architecture optimization methods: A systematic literature review, Soft. Eng., IEEE Trans. 39 (5) (2013) 658 – 683.

[45] M. Drago, C. Ghezzi, R. Mirandola, A quality driven extension to the QVT-relations transformation language, Computer Science - Research and Development 30 (1) (2015) 1–20.

[46] M. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, J. Merseguer, Performance by unified model analysis (puma), in: WOSP 2005, 2005.

[47] T. Parsons, J. Murphy, Detecting performance antipatterns in component based enterprise systems, Journal of Object Technology 7 (3) (2008) 55–91.

[48] C. Smith, L. Williams, More new software performance antipatterns: Even more ways to shoot yourself in the foot, in: CMG 2003, 2003.

[49] A. Kavimandan, A. Gokhale, Applying model transformations to optimizing real-time qos configurations in dre systems, Architectures for Adaptive Software Systems (2009) 18–35.

[50] G. Loniewski, E. Borde, E. Insfran, Towards a model driven refinement process through architecture evaluation, in: Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages, NFPinDSML 2012, ACM, New York, NY, USA, 2012, pp. 4:1–4:6.

[51] R. Li, R. Etemaadi, M. T. M. Emmerich, M. R. V. Chaudron, An evolutionary multiobjective optimization approach to component-based software architecture design, in: CEC 2011, 2011.

[52] A. Aleti, S. Stefan Björnander, L. Grunske, I. Meedeniya, Archeopterix: An extendable tool for architecture optimization of aadl models, in: MOMPES 2009, 2009.

[53] I. Meedeniya, B. Buhnova, A. Aleti, L. Grunske, Architecture-driven reliability and energy optimization for complex embedded systems, in: QoSA 2010, 2010.

[54] G. Canfora, M. Di Penta, R. Esposito, M. Villani, An approach for qos-aware service composition based on genetic algorithms, in: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM, 2005, pp. 1069–1075.

[55] M. Ouzineb, M. Nourelfath, M. Gendreau, Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems, Reliability Engineering & System Safety 93 (8) (2008) 1257–1272.

[56] E. Bondarev, M. R. V. Chaudron, E. A. de Kock, Exploring performance trade-offs of a jpeg decoder using the deepcompass framework, in: WOSP 2007, 2007.

[57] B. Eames, S. Neema, R. Saraswat, Desertfd: a finite-domain constraint based tool for design space exploration, Design Automation for Embedded Systems 14 (1) (2010) 43–74.

[58] T. Saxena, G. Karsai, Mde-based approach for generalizing design space exploration, Model Driven Engineering Languages and Systems (2010) 46–60.

[59] E. Jackson, E. Kang, M. Dahlweid, D. Seifert, T. Santen, Components, platforms and possibilities: towards generic automation for mda, in: Proceedings of the tenth ACM international conference on Embedded software, ACM, 2010, pp. 39–48.