Architectural Design of Cloud Applications: a QoS-aware Cost Minimization Approach

Michele Ciavotta, Giovanni Paolo Gibilisco, Danilo Ardagna, Elisabetta Di Nitto and Marcos Aurélio Almeida da Silva.

Abstract—Cloud Computing has assumed a relevant role in the ICT, profoundly influencing the life-cycle of modern applications in the manner they are designed, developed, and operated. The Cloud market offers highly diversified services upon which developers and operators can rely. Yet, its full adoption requires specific and rare expertise. Actually, such services are characterized by a steep learning curve and, among the other aspects, by significantly different Quality of Service (QoS). In this paper, we tackle the problem of supporting the design-time performance analysis of Cloud applications and the identification of the optimal strategy for allocating components onto Cloud resources. In particular, the final goal is to set the basis to overcome the limitations of current design-alternatives search tools proposing (i) a mathematical formulation for the underlying optimization problem, i.e., determine the Cloud configuration that minimizes the execution costs of the application over a daily time horizon, fulfilling at once QoS and service allocation constraints, and (ii) a software solution able to efficiently solve the resource allocation problem. The tool, codenamed SPACE4Cloud, embodies a hybrid two-level search meta-heuristic for the efficient exploration of the design-alternatives space. The benefits of this approach are demonstrated in the context of an industrial case study. Furthermore, this work also presents results where SPACE4Cloud leads to a cost reduction up to 60%, when compared to a first-principle technique based on utilization thresholds, like the ones typically used in practice. Finally, an extensive scalability analysis indicates that our solution is able to solve large problem instances within a time frame compatible with a fast-paced design process (less than half an hour in the worst case).

Index Terms—Model-Driven Software Development, Search-based Software Engineering, Performance Assessment, Cloud Computing, Cost Minimization, Quality of Service

1 Introduction

Undeniably, building an application in a Cloud Computing context showcases significant advantages as it offers the possibility to rely upon a wide range of preexisting battle-tested, highly-available, and fully managed services. The cloud offering covers the whole technological stack from virtual machines/containers and storage to Databases as a Service, to middleware components like message queues, and to complete Software as a Service solutions. Nonetheless, the downside is that the adoption of such services is not painless as it may seem. Among the other issues, these services show different Quality of Service (QoS) characteristics and cost, which inevitably impact the overall performance and cost of the final application.

This paper focuses on supporting the performance analysis of cloud applications and the identification of an optimized strategy for allocating application components onto cloud resources.

In general, the problem of architectural optimization is deemed especially difficult to solve. As Koziolek, Koziolek and Reussner state in [1], due to the increasing size and complexity of software systems, architects have to choose from a combinatorially growing number of design options when seeking for an optimized architecture that, in a traditional in-house scenario, takes the allocation to the physical resources into account. In this context, the literature provides methods for automated design-space exploration (see for instance [2], [3], [4]) and high-level models and tools to support the software architect (see, e.g., *Palladio Component Model* and its *Palladio Bench*, and *PerOpteryx* design environment [5], [1], or stochastic process algebra and *PEPA Eclipse plugin* [6], [7]). Such methods and tools focus on identifying the best architectural configuration given a set of QoS requirements, but they do not support cloud-specific abstractions and do not directly address the problem of deriving an optimized cloud configuration.

Arguably, the problem becomes significantly more complex when the design process is broadened in its degrees of freedom by the possibility to choose among several alternative cloud services [8]. As a matter of fact, the service selection problem has been traditionally considered separately from the software architecture design. The possibility to exploit cloud services, presently offered in high numbers, each impacting differently the overall application behavior and execution costs, makes the selection and allocation tasks particularly cumbersome. Trivially, consider the case of a single-tier web application executed by a server deployed on top of a Virtual Machine (VM) service offered by Amazon. In order to deploy such a simple application, one has to decide the size of the VM (hereinafter also referred to as *type*) as well as the number of its replicas. Picking a few instances of a powerful VM type like the *c4.4xlarge* or a higher number of instances of a less powerful VM type, like the *m5.large*, can have a significant effect both in terms of cost and performance. As discussed in [9] making the wrong decision can significantly degrade performance and increase applications operational cost by 2-3x on average, and about 12x in the worst-case. Therefore, this work proposes an approach wherein the designer firstly defines the high-level architecture of his/her application and then moves to a step where he/she allocates cloud services to that architecture and assesses the validity of the obtained configuration based on some predefined QoS constraints. This step should precede the actual coding phase, as part of the architectural design phase; some of the

- *M. Ciavotta is with Dipartimento di Informatica, Sistemistica e Comunicazione, Università di Milano-Bicocca, Italy. E-mail: michele.ciavotta@unimib.it*
- *G.P. Gibilisco, D. Ardagna, and E. Di Nitto are with Dipartimento di Elettronica, Informaione e Bioingegneria, Politecnico di Milano, Italy. E-mail: name.lastname@polimi.it*
- *M.A. Almeida da Silva is with Softeam, Paris, France*

Manuscript received XXX XXX XXX; revised XXX XXX XXX.

choices may, in fact, entail specific technological constraints on the development of the application components. Constraints may concern, for instance, the use of a specific programming language as a consequence of the selection of a particular container service, or the use of a specific API forced by the choice of a particular Database as a Service.

Evaluating the cost-performance trade-off of a component to be deployed on a specific cloud resource (throughout this paper the terms *service* and *resource* are used interchangeably), possibly before that component is developed, it is undoubtedly a non-trivial task, all the more so for an entire application. Therefore, as part of this work, we propose an automated tool, namely *SPACE4Cloud*, capable of deriving quantitative (performance and cost) information on the deployment in the early stages of the design process. Having quantitative information available at design-time will enable designers to make informed decisions about the technology to use and the architecture to implement to fully exploit the potentiality the cloud offers. More concretely, SPACE4Cloud is able to determine, through an efficient generation/assessment of configuration alternatives, the cloud deployment that minimizes the resource usage costs while fulfilling *QoS* and *service allocation constraints*. The latter restrict the possible deployment configurations to decisions made by the designer, imposing conditions on the type and characteristics of services to be selected. For instance, possible allocation constraints for a compute resource might set the minimum or the maximum number of cores or the available memory. Our solution, differently from most of those available in the literature, is specially devised to tackle cloud-based systems, which are intrinsically dynamic. In particular, it considers resource performance, price, and the incoming workload as subject to fluctuation over time. For this reason, our approach introduces a time-dependent workload profile over a 24-hour time horizon (which leads to the problem of solving 24 intertwined capacity allocation problems) and exploits a statistical characterization (via Random Environment [10]) suitable to capture cloud multi-tenancy contention and the resulting performance variability of services. Specifically, SPACE4Cloud models the cost optimization problem of an application to be deployed in a cloud environment as structured in two decision levels. The topmost decision level involves assigning the most appropriate virtual resource type (namely, IaaS Virtual Machines), selected from the set of those that meet the service allocation constraints, to each application tier. The problem faced at the lower level is to estimate, for each hour of the day and each application tier, the minimum number of resources that guarantees the overall application to exhibit the QoS levels established and minimizes the daily execution costs. The resulting optimization problem is $\mathcal{NP}$-hard and is solved by a hybrid two-level search-based meta-heuristic.

Compared to previous literature proposals [1], [11], [12], [13], SPACE4Cloud generates more robust solutions as it considers the time-dependent nature of the parameters relevant to the analysis and, in particular, of the workload. Other approaches, instead, perform capacity allocation only at the workload peaks. Moreover, SPACE4Cloud outperforms the other approaches also in terms of time required to perform the optimization. In fact, while the others tend to require processing times in the range of hours to provide a solution [13], [14], SPACE4Cloud demonstrated to be effective and efficient, providing solutions in the span of minutes (even if, considering the time-varying workload profile, it solves 24 optimization problem instances instead of a single one). Finally, when compared with first-principle techniques based on utilization thresholds [15], [16] (typically used in practice and implemented by cloud providers, see, e.g., Amazon AWS Elastic Beanstalk[1]) our solution enables the designers to generate architecture configurations that lead to a cost reduction up to 60%. This paper also reports on the application of SPACE4Cloud to a real industrial case study where the tool has adequately supported our industrial partner (Softeam) in evolving the architecture of their cloud application Modelio[2] by identifying an alternative deployment that proved to be both more scalable with respect to the one devised initially and with lower operational costs.

SPACE4Cloud is available[3] under the open source Apache 2.0 license. The tool is presented in detail, including the internal components, workflow and bi-level optimization algorithm, in this work for the first time; nevertheless, references to it can be found in [17] where an approximate mixed integer linear formulation for the considered problem is presented and discussed (SPACE4Cloud exploits that formulation to swiftly identify an initial solution) and in [14] where our tool is proposed in combination with PerOpteryx [1] to provide optimization capability over 24-hour time horizon and the management of more advanced constraints (i.e., on the response time percentiles).

In the rest of this paper the case study used to motivate our work (Section 2) is introduced. Then in Sections 3 and 4 the SPACE4Cloud design approach and its architecture is presented. Section 5 focuses on the formulation of the optimization problem and on the design-time exploration algorithm. In Section 6 the approach is evaluated in the industrial use case scenario and against first principle heuristics. Moreover, a rigorous scalability analysis is reported. Finally, Section 7 discusses other proposals available in the literature whereas Section 8 concludes the paper.

2 The Modelio Case Study

Modelio is a software solution that provides support to the most widely used modeling and methodology standards. It is available in an open source version as well as and in a family of enterprise licenses. It is a comprehensive Model Driven Engineering (MDE) workbench featuring a central IDE that allows various languages (represented as UML profiles) to be combined in the same model.

The standard version of Modelio is a desktop application; Softeam decided to extend Modelio's features and architecture to enable sharing projects and models among multiple stakeholders and to provide other prime functionalities like version control and access policies. The result of this work is a cloud-based extension of Modelio called *Constellation*, which has been recently delivered as a service[4]. Constellation uses a central repository, implemented by a Subversion[5] (SVN) server, for model governance and administration. Users interact with the repository by committing and retrieving changes of the shared models. Additionally, the SVN server supplies user authentication and authorization, and provides a resolution model for conflicts that may arise when multiple users modify the same model simultaneously.

In a traditional setup, users that acquire Constellation for in-house deployment would need to provision their infrastructure and secure the server, which in turn would entail investments for hardware and maintenance. To facilitate the adoption, Softeam

1. http://aws.amazon.com/elasticbeanstalk/
2. https://www.modelio.org
3. https://github.com/deib-polimi/modaclouds-space4cloud
4. https://www.modeliosoft.com/en/products/modelio-constellation.html
5. https://subversion.apache.org

decided to expand its service portfolio relieving its customers of this burden by providing a Constellation as a cloud hosting and management service. This choice appeared necessary as most of Softeam customers are SMEs and would not be able to benefit from Constellation if this had implied a significant management effort. At the same time, such a decision entailed a significant change in the company's strategies. In fact, while traditionally Softeam's core activity has been exclusively focusing on developing a single-user desktop product, now it is moving to a completely different context in which not only it has to deal with the design and development of a multi-tenant system, but it also has to operate this system keeping operational costs under control. This translates into the following requirements:

- The company should be able to study the feasibility of different cloud deployments subject both to QoS constraints as well as to the specific customers' needs.
- The cloud operations costs should be kept under control in order to ensure the company to achieve the shortest possible payback period.
- Finally, the company wants to be able to set specific QoS constraints on the components of the Constellation architecture and wants to exploit the cloud elasticity to cope with the time-dependent number of connected users and the related workload.

In the next sections, SPACE4Cloud is presented, showing how it addresses these concerns, thus helping Softeam in accomplishing its main objectives.

3 The SPACE4Cloud Design and Analysis Approach

Figure 1 shows the modeling workflow supported by the SPACE4Cloud approach. The first phases concern modeling the structure of the application in terms of its deployable components and of its QoS requirements and constraints, as well as the workload expected for such application. Such workload, in many cases, is not constant and varies throughout the day. When these models are completed, the computation of the cloud services optimal allocation can start. In this phase, the models from the previous two phases are analyzed in combination with the service models of the candidate cloud resources. These are stored in the Repository of Cloud Services.

A service model is a representation of those attributes of a cloud service that are relevant to the assessment of its QoS. For instance, in this context, a virtual machine (VM) is modeled in terms of CPU speed and number of cores, and memory available. The definition of such traits is not necessarily an easy task as it requires information to be collected from various sources and, whenever data is not available, it involves a benchmarking process [18], [19], [20]. On that account, in the SPACE4Cloud workflow, this information is assumed to be provided by experts and maintained in the Repository of cloud services. At the time of writing the repository[6] contains models of services by Amazon AWS, Microsoft Azure, Flexiscale, ProfitBricks, and CloudSigma.

The final result achieved by the workflow is the assignment of the application components to cloud resources, taking into account all requirements and constraints and minimizing the cost of resources. As such an assignment might be unsatisfactory,

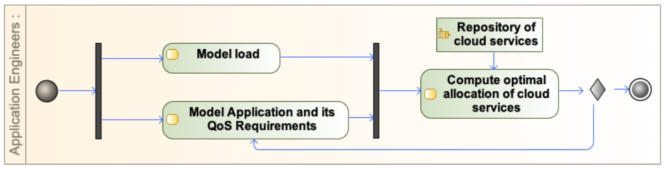

Application Engineers :
Model load
Model Application and its QoS Requirements
Repository of cloud services
Compute optimal allocation of cloud services

Fig. 1: Main modeling steps in the SPACE4Cloud approach.

the reiteration of all previous phases is enabled. The following subsections describe the activities of the modeling workflow in greater detail.

3.1 Modeling the application

Application modeling includes various aspects ranging from framing the architectural components to defining the QoS and service allocation constraints associated with each component. The language adopted for all modeling activities is *MODACloudML* [21], [22], [23], which is structured as a set of Domain Specific Languages (DSLs) that allow the application designer to include all pieces of information that are needed for the execution of the architectural analysis and optimization process [24]. MODACloudML is supported by Creator4Clouds[7], an open source modeling IDE for Cloud applications and QoS modeling.

Figure 2 shows the MODACloudML diagram describing the main components of the *Constellation* architecture and their connections. They are: an *Administration Server*, coupled with an *Administration Database*, to keep track of users, profiles, projects and resources, and a set of *Agents*. The *SVN Agent* hosts the customers' UML work models; the *HTTP Agent* provides read-only access to UML models to be shared among various projects. Finally, the *Agent Manager* coordinates the agents, distributing among them resource-intensive tasks, such as model transformations, document generation, among the others.

The activity that completes the architectural modeling phase is the definition of the *demanding times* (i.e., the distribution of times required to execute a single request without contention on a reference system [25]) of the operations offered by *Constellation* [26]. This step is essential to perform a sound performance analysis of the modeled application. The demanding times can either obtained from an educated guess based on experts' experience or derived from monitoring a prototype implementation and relying on state-of-the-art parameter estimation techniques [27], [28]. As regards *Constellation*, the second approach has been adopted; specifically, the response times of prototype components deployed on an Amazon EC2 Medium instance have been recorded. Notice that, in order to be able to assess the application behavior on different configurations, the times gathered experimentally have to be normalized to phase out the dependency on the particular benchmark instance. The resulting demanding times are reported in Table 1.

Figure 2 also reports some constraints on Constellation components and operations that specify desired QoS levels. In particular, the expected time to read a model from the SVN server should be at most 15 seconds on average. Besides, since committing a model requires more complex processing since it might include conflict detection and resolution, the expected average response time for the commit functionality is set to 60 seconds. For the HTTP server

6. https://github.com/deib-polimi/modaclouds-resourcemodel

7. http://forge.modelio.org/projects/creator-4clouds

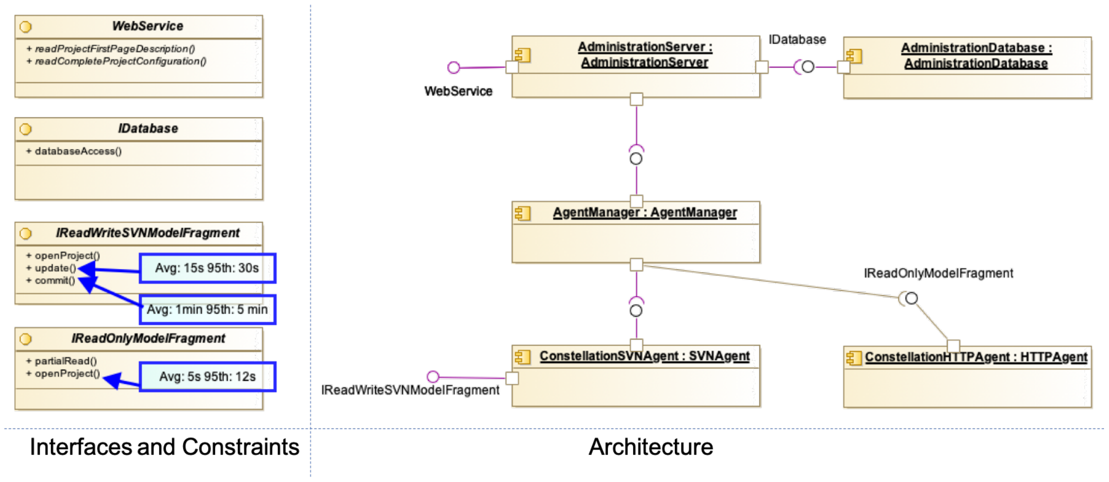

WebService
+ readProjectFirstPageDescription()
+ readCompleteProjectConfiguration()
IDatabase
+ databaseAccess()
IReadWriteSVNModelFragment
+ openProject()
+ update()
+ commit()
Avg: 15s 95th: 30s
Avg: 1min 95th: 5 min
IReadOnlyModelFragment
+ partialRead()
+ openProject()
Avg: 5s 95th: 12s
Interfaces and Constraints
AdministrationServer : AdministrationServer
IDatabase
AdministrationDatabase : AdministrationDatabase
WebService
AgentManager : AgentManager
IReadOnlyModelFragment
ConstellationSVNAgent : SVNAgent
ConstellationHTTPAgent : HTTPAgent
IReadWriteSVNModelFragment
Architecture

Fig. 2: Overview of Constellation architecture, interfaces and constraints.

Operation	Average time (ms)
readProjectFirstPageDescription	597
readCompleteProjectConfiguration	723
databaseAccess	900
openProject (SVN)	15,000
update (SVN)	2,500
commit (SVN)	41,000
partialRead (HTTP)	1,000

TABLE 1: Constellation demanding times.

hosting partial, read-only models, the average time for reading a model is required to be lower than 5 seconds. Besides constraints on the average response time, constraints on the 95th percentile of the response time are also defined. This has been done in order to obtain a fine-grained control on the maximum acceptable response time for critical functionalities. In the Constellation example, these constraints are set to 12 seconds for HTTP reads, 30 seconds for SVN reads and 5 minutes for commits.

Ultimately, the designer can specify requirements in the form of *service allocation constraints*, which predicate on the resources considered for the deployment with the aim of narrowing the space of possible configurations. These constraints can also be used to define minimum requirements that cloud resources have to meet in order to host critical components of the system, like the minimum amount of cores and memory needed to host a database component. For the case of Constellation, technological considerations impose that the Administration Server, Administration Database, Agent Manager and the SVN Agent cannot be replicated. As a consequence, the tiers hosting these components can not scale (i.e., the number of VMs for these tiers is fixed to 1). Furthermore, to avoid instantiating these components on cloud resources that would not guarantee the required levels of QoS, the Constellation designer has imposed that no component can be deployed on a VM type featuring less than 2GB of memory, whereas for the SVN Agent, which performs more time-demanding operations, the minimum requirement is set to 8GB.

3.2 Modeling the expected load

In order to identify the proper resource capacity for the application under study, the designer needs to provide a model of how the application is expected to be used during its normal operation. Such information is provided in terms of *activity diagrams* describing the interactions between users and application along with the expected probability of occurrence (*user behavior*, an extension provided by MODACloudML to traditional UML activity diagrams). For instance, referring to the *Constellation* case, the user behavior is depicted in Figure 3. It shows how one or multiple teams of users access *Constellation* to work on shared models. The user behavior has been built by logging the accesses in the current Modelio implementation as well as by analyzing common SVN usage patterns. From these analyses, it results that the users perform preferably model reads, while updates are less frequent as, fundamentally, users work on models using the Modelio client installed on their own machines and commit the updates on the shared models only a few times per day. The number on the top left corner of each branch in the diagram identifies its likelihood. In our example, 5% of requests corresponds to users accessing a project to obtain its related information while 10% of time users retrieve model updates or commit changes to the SVN server. The rest of the time, users access read-only models shared by means of HTTP fragments.

Besides the behavior of the various classes of users, the designer has to specify also the expected traffic in terms of the number of users that hits the application during the operational time interval. A typical interval is in our view 24 hours, which is meaningful under the assumption that the traffic shows similar patterns in different days. Taking into consideration the full characterization of workload in the 24 hour time period (*reference day*) allows SPACE4Cloud to derive different deployment configurations, one for each hour of the day, thus enabling the possibility to adapt the size of the infrastructure to the incoming workload by exploiting the cloud elasticity [13]. Note that, most of the cloud providers nowadays adopt a hourly billing model[8][9].

For what concerns the case study, it has been used the load distribution observed in the current installations that shows a daily pattern with a bi-modal distribution and an average think time of 10s. Such workload, suitably scaled, is also used in Section 6.1 to evaluate the scalability of the considered architecture.

8. http://aws.amazon.com/ec2/pricing
9. http://www.windowsazure.com/en-us/pricing/details

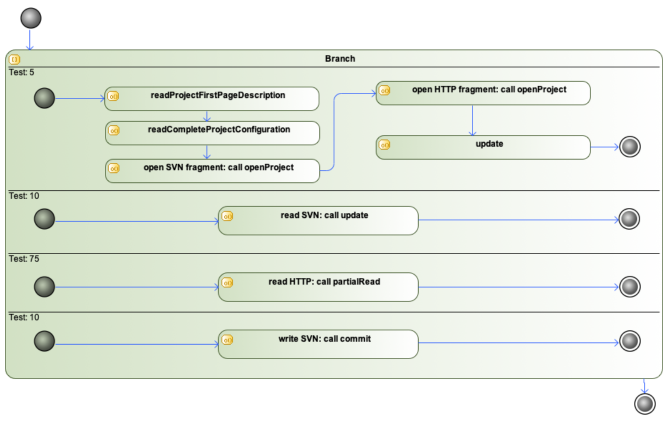

Branch
Test: 5
readProjectFirstPageDescription
readCompleteProjectConfiguration
open SVN fragment: call openProject
open HTTP fragment: call openProject
update
Test: 10
read SVN: call update
Test: 75
read HTTP: call partialRead
Test: 10
write SVN: call commit

Fig. 3: Overview of Constellation user behavior.

3.3 Optimizing the Application Deployment

This phase is semi-automated by the SPACE4Cloud tool, provided that all other activities have produced the corresponding results. In particular, the tool inputs are *i)* the repository of cloud services; *ii)* the application architecture, together with the information about the demand of each component, and the QoS and Service Allocation constraints; and *iii)* the model of the expected users' behavior. These models, presented in the previous sections, can be created using MODACloudML in the Creator4Cloud IDE and exported in the format used by SPACE4Cloud, which is an extension of the well-known Palladium Component Model (PCM) [5], referred to as Extended PCM models in the rest of the paper. This choice enabled us to exploit a broad set of tools for QoS analysis developed in other contexts as well as to use SPACE4Cloud in combination with other solutions [14]. Details on the SPACE4Cloud tool and on the optimization process are reported in Sections 4 and 5.

The outcome of this phase is a set of assignments of application components to cloud services covering the entire time horizon, and the minimum level of replication of these services required to guarantee the desired QoS for each hour.

3.4 Reiteration

Modeling and analyzing the architecture of a cloud application should be considered as an iterative process. In fact, the optimization phase may provide unsatisfactory results that may suggest designers adopt different architectural patterns or to alter some other elements such as the allocation constraints or the expected workload. For instance, in the Constellation case, the result of the optimization phase, which highlighted a limit of the offered solution in terms of its scalability with the growth of contemporary users, has encouraged the company to identify a different architectural alternative as discussed in Section 6.1.2.

4 SPACE4Cloud: The Optimization Tool

SPACE4Cloud implements an efficient exploration of the design-alternatives space that seeks the cloud deployment that minimizes the resources leasing costs while fulfilling QoS and service allocation constraints over the reference time horizon. To accurately assess the performance of the application under development (and, equivalently, to provide QoS guarantees at design time), SPACE4Cloud translates the design models (videlicet the PCM models extended with the time-variant workload and Random Environments [24]), convenient for easy editing, into performance models, suitable for automated analysis. The input PCM models are, therefore, converted into Layered Queueing Networks (LQNs) [26] and processed by a solver. The specific LQN solver employed in this work is LINE [29], which is currently the only capable of managing Random Environments and evaluating response time percentiles. SPACE4Cloud, however, also supports the LQNS solver [30] (clearly at the cost of sacrificing percentiles and Random Environments) and can perform availability evaluations via the PCM Reliability Solver [31].

LQNs have been preferred over other performance models as they allow to represent complex systems (e.g., multi-tier applications) and resource competition at the software (and not only hardware) level. Since proving the appropriateness of LQNs to model cloud solutions is beyond the objective of this study, the interested reader can refer to [29], [32], [33]; noteworthy, however, is that [33] estimates the gap between the QoS prediction using LQN and the average response time measured in a cloud deployment under precise conditions to be as small as 2%.

Figure 4 highlights the main components of SPACE4Cloud as well as the links with third-party components. The *Initial Solution Builder* derives an initial allocation for the application under analysis by solving a Mixed Integer Linear Program (MILP) based on M/G/1 queue models (these are less accurate than LQNs but can be evaluated through closed formulae). This initial solution is generated to be biased toward promising regions of the search space and, thus, improves the performance of the metaheuristic algorithm executed within the *Tabu-search Optimizer* component (see [12] for another similar approach). Notice that, the rigorous formulation of the problem, the optimization algorithm, and their accurate evaluation constitute the innovative core of this research whereas the formalization of the MILP problem (that is, ultimately, a particular instance of a formulation, similar to the one proposed here and yet weaker, as it does not cover the constraints on the percentiles) as well as the impact of its adoption on the overall optimization process is not discussed in this paper as it has already been presented and analyzed in [17].

The *MILP Solver Connector* decouples SPACE4Cloud from the specific solver adopted for the MILP. All experiments reported in this paper have been conducted using CPLEX[10] MILP solver; nevertheless, alternative solutions, e.g., the open source GLPK[11], are supported as well.

The *Tabu-search Optimizer* component implements a local search exploration procedure designed to improve the current best solution iteratively. This method draws its inspiration from both Tabu [34] and GRASP [35] paradigms and is detailed in Section 5.3. Concretely, the algorithm generates a new solution (the *candidate* solution) iteratively by applying small transformations (*moves*) to the current best solution, evaluates it in terms of feasibility and cost, and if the candidate solution is better than the current best, the latter is replaced. A candidate solution for the problem under analysis is composed of 24 allocation configurations, one for each hour of the day. Each of these leads to a different LQN model to be evaluated. The evaluation is a time-demanding task and the main bottleneck for the overall optimization process as it entails two phases, in the first one each LQN model is analyzed by the external LQN solver to evaluate a number of performance indexes that are, in turn, compared with the QoS

10. http://www.ibm.com/software/commerce/optimization/cplex-optimizer/
11. http://www.gnu.org/software/glpk/glpk.html

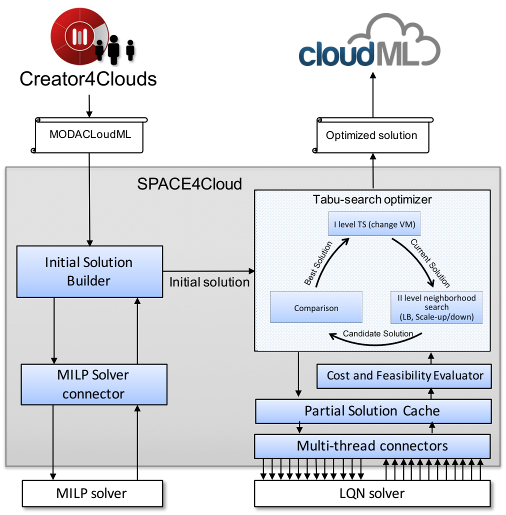


Fig. 4: SPACE4Cloud architecture.

constraints to assess the feasibility of the overall solution. Second, the tool evaluates the cost of the candidate solution according to the cost model presented in [24], where an hourly pricing model is considered for cloud resources. For the proposed tool to be usable to solve real-life problems, the evaluation process had to be accelerated by means of a multi-thread connector component managing the parallel evaluation of the 24 LQN models of a single solution and a cache-based proxy (*Partial Solution Cache*) to store the evaluations of previous solutions for each hour in the considered time horizon. These additions have significantly boosted the optimization process since, usually, the Tabu-search optimizer generates similar solutions; thus, by caching partial evaluations, the tool can avoid unnecessary executions of the underlying LQN solver.

Finally, the optimized configuration found can be input to solutions such as CloudML [21], [36] for the automation of the deployment process in the selected cloud.

5 Design-Time Architecture Optimization

This section presents the principles behind the *Tabu-search optimizer* component of the SPACE4Cloud architecture. In particular, in Section 5.1 the problem under analysis is defined. Section 5.2 provides an analytical formulation for the problem, characterizing objectives and constraints. Finally, in Section 5.3 the optimization algorithm is outlined.

5.1 Problem definition

As discussed in Section 2, an application is built from several components, each of them implementing a set of functionalities. Components (e.g., the Constellation's Administration Server) are grouped in tiers, each allocated on a pool of resources (as VMs); generally, tiers can scale to handle dynamic changes in the application environment such as fluctuations in the incoming workload. The resources allocated to different tiers are assumed to be heterogeneous whereas those exploited within the same tier are homogeneous. This is not restrictive as it is compliant with auto-scaling mechanisms offered by current IaaS platforms. In fact, the management of heterogeneous resources in the same tier is not considered applicable in public clouds [37], [38], as it complicates the resource management process and it could even degrade application performance in case of unbalanced workload [39].

The goal is to find the cheapest configuration (in terms of resource type and number of replicas for each application tier) able to fulfill *QoS* and *service allocation constraints* for every hour of the *reference day*. As discussed previously, *service allocation constraints* allow to predicate on the characteristics of the infrastructure and can include minimum requirements to host application components (e.g., minimum amount of RAM, minimum and maximum number of replicas for a tier, maximum CPU utilization, etc.). *QoS constraints* allow to predicate on the service level experienced by the end user (e.g., 95% of *commit* requests to be served within 5 min). For the sake of simplicity, in the following only QoS constraints predicating on functionality-level response time and service allocation constraints predicating on memory size are formalized. Other allocation as well as QoS constraints on other performance metrics (e.g., throughput) can be introduced similarly.

5.2 Problem Formulation

As said, an application is hosted on a set of tiers, denoted by $\mathcal{I}$, that supports the execution of application components. Each tier consists of multiple homogeneous resources sharing evenly the incoming workload. Let $\mathcal{V}$ be the set of available types of resources (e.g, Amazon EC2 m5.large[12] or Azure Standard_F2s_v2[13]) and $\mathcal{T}$ a set defined by the N time intervals in which the reference period is split (i.e., 24 time intervals of one hour if a day is considered as reference). Each resource type $v \in \mathcal{V}$ is characterized by the corresponding model stored in the *Repository of cloud services*. In particular, the following parameters are relevant for the optimization: price $C_{v,t}$, variable over the time horizon, speed S_v and memory size M_v, which can be derived from the processing rate and cost profiles available on the provider's catalogs.

The workload is described in terms of number of users (population) π_t, with $t \in \mathcal{T}$, characterized by a certain think time Z (namely, the average time between the instant a user receives a response from the system and the one he/she submits a new request [26]). Each user interacts with the application executing a request according to the defined user behavior model, like the one in Figure 3; the set of possible requests is denoted by $\mathcal{K}$. Each request $k \in \mathcal{K}$, in turn, is characterized by a probability to be executed and by an ordered list of components supporting its execution (i.e., its *execution path* [40]). As shown in Figure 5, in Section 6.1, SPACE4Cloud supports the definition of QoS constraints in terms of thresholds on the average response time *Avg* and on the percentiles *Perc*.

With $\mathcal{K}_{Avg} \subseteq \mathcal{K}$ and $\mathcal{K}_{Perc} \subseteq \mathcal{K}$ are denoted the subsets of requests on which are defined constraints on the average times and on the percentiles of the response time, respectively. Consequently, the set of QoS constraints predicating on the average response time (R_k) can be expressed as $E(R_k) \leq \overline{R}_k^E \quad \forall k \in \mathcal{K}_{Avg}$ where E is the expectation and $\overline{R}_k^E$ is a threshold. Similarly, let $P(R_k \leq \overline{R}_k^P)$ be the cumulative distribution function for the response time R_k representing the probability that R_k takes values below a threshold $\overline{R}_k^P$, the inequalities $P(R_k \leq \overline{R}_k^P) \geq \alpha_k \quad \forall k \in \mathcal{K}_{Perc}$ denote the requirements that at least a percentage of k-functionality response times (α_k-th percentile) falls in the interval $[0, \overline{R}_k^P]$.

Since the objective of the problem is to select for each application tier both the most suitable resource type and the number

12. http://aws.amazon.com/ec2/instance-types
13. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-compute

of replicas of that resource over the time horizon, the decision variables associated with the formulation are accordingly:

- $w_{i,v}$, which takes value 1 if the resource type v is assigned to the i-th tier, 0 otherwise;
- $z_{i,v,t}$, which specifies the number of resources of v type (replicas) assigned to the i-th tier at time t.

Finally, the set of decision variables $w_{i,v}$ is denoted by $\mathcal{W}$ while $\mathcal{Z}_t$ represents the set of decision variables $z_{i,v,t}$ at time t. The complete lists of the model parameters and variables are summarized in Tables 2 and 3, respectively.

It is worth noticing that the response time R_k^t is at every time interval t function of the workload (number of users, think time), resource types and number of replicas, i.e. $R_k^t = R_k^t(\pi_t, Z, \mathcal{Z}_t, \mathcal{W})$. As presented and discussed in [17] R_k^t can be approximated by M/G/1 (as in the models solved by Initial Solution Builder) or estimated using LQN models and simulators [30], [26] (as in the Tabu-search Optimizer), thus for the sake of generality R_k^t is not detailed further.

Given these premises, the optimization problem (P) for the sake of clarity is formulated as a bi-level optimization problem featuring an *upper-level problem*, dealing with the selection of the resource type for each tier, and a *lower-level problem* related to the assignment of the most suitable number of resources to each tier. More formally, we have

$$(P) \qquad \min \sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} \sum_{t\in\mathcal{T}} C_{v,t} z_{i,v,t} \tag{1}$$

Subject to:

$$\sum_{v} w_{i,v} = 1 \qquad \forall i \in \mathcal{I} \tag{2}$$

$$\sum_{v} M_v w_{i,v} \geq \overline{M}_i \qquad \forall i \in \mathcal{I} \tag{3}$$

$$w_{i,v} \in \{0,1\} \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V} \tag{4}$$

$$z_{i,v,t} = \arg\min_{\mathcal{Z}} \sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} \sum_{t\in\mathcal{T}} C_{v,t} z_{i,v,t} \tag{5}$$

Subject to:

$$E[R_k^t(\pi_t, Z, \mathcal{Z}_t, \mathcal{W})] \leq \overline{R}_k^E \qquad \forall k \in \mathcal{K}_{Avg}, \forall t \in \mathcal{T} \tag{6}$$

$$P(R_k^t(\pi_t, Z, \mathcal{Z}_t, \mathcal{W}) \leq \overline{R}_k^P) \geq \alpha_k \qquad \forall k \in \mathcal{K}_{Perc}, \forall t \in \mathcal{T} \tag{7}$$

$$z_{i,v,t} \geq w_{i,v} \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T} \tag{8}$$

$$z_{i,v,t} \leq \Theta w_{i,v} \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T} \tag{9}$$

$$z_{i,v,t} \;\; Integer \qquad \forall i \in \mathcal{I}, \forall v \in \mathcal{V}, \forall t \in \mathcal{T} \tag{10}$$

Where $\mathcal{T} = \{1 \dots N\}$ and $\mathcal{Z} = \{z_{i,v,t} | (i,v,t) \in \mathcal{I} \times \mathcal{V} \times \mathcal{T}\}$.

Note that, as implemented in most public clouds (e.g., Amazon Autoscaling Groups[14]), the formulation assumes that during the execution of the application, only horizontal scaling can be performed (i.e., the number of replicas is increased or decreased keeping untouched the type). For this reason, the number of replicas provisioned for each tier $z_{i,v,t}$ (optimized in the lower-level models) depends on the time of the day t, while the resource type $w_{i,v}$ (optimized in the upper-level model) does not. This assumption eases the configuration process, since fine tuning the configuration of application components for a certain virtual resource type (in terms of number of open threads or heap size, etc.) is a cumbersome task and repeating this process for all of the types of machines offered by cloud providers would require an effort for the software architect that can be safely stated impracticable.

14. https://aws.amazon.com/it/autoscaling

Provided that $w_{i,v}$ are binary variables, condition (2) guarantees that exactly one type of resource can be selected for each tier. Condition (8) in combination with (2) and (4), in turn, guarantees that a non-empty set of resources is assigned to each tier. Moreover, condition (9) imposes each tier to be composed by homogeneous resource; if $w_{i,v} = 0$ the total number of resources of type v assigned to tier i is forced to zero and this occurs for all $v \in \mathcal{V}$ but one (2). Besides, if $w_{i,v} = 1$ the number of replicas assigned to tier i is at least 1 (8) (Θ is an arbitrary large integer number). Equation (3) ensures that the resource of type v selected to host tier i fulfills any constraint on the minimum amount of memory $\overline{M}_i$ specified for that tier. Finally, equations (6)-(7) represent the constraints on the average and the percentile of response times, respectively.

In the light of the considerations made so far, the expression $\sum_{v\in\mathcal{V}} C_{v,t} z_{i,v,t}$ represents the cost of the resources assigned to tier i at time t, while $\sum_{i\in\mathcal{I}} \sum_{v\in\mathcal{V}} C_{v,t} z_{i,v,t}$ is the cost of the whole application at time t. Therefore, the total cost of the considered application over the 24 hours can be calculated summing this latter value over every time interval. The result of this operation is the objective function in (1). Ultimately, notice that (P) can also be formulated in a more compact fashion as (in the more general case where $E(R_k)$ and $P(R_k)$ are non-linear) a Mixed Integer Non-Linear Problem, which is $\mathcal{NP}$-hard as it is a generalization of the Mixed Integer Linear Problem, proven to be $\mathcal{NP}$-hard in [41].

Problem parameters

Index	
t	time interval
i	tier
k	functionality
v	type of virtual resource
Parameters	
π_t	number of concurrent users at time t
Z	users think time
$C_{v,t}$	cost of a single replica of resource of type v at time t
S_v	speed of a resource of type v
M_v	memory of a resource of type v
$\overline{M}_i$	memory constraint for tier i
$\overline{R}_k^E$	maximum average response time for the k-functionality
α_k	minimum response time percentile value for the k-functionality
$\overline{R}_k^P$	response time value representing a percentile ($\geq \alpha_k$-th) of the response time distribution function associated with the k-functionality

TABLE 2: Optimization model parameters.

Optimization model decision variables.

$w_{i,v}$	binary variable that is equal to 1 if the VMs type v is assigned to the i-th tier and equal to 0 otherwise
$z_{i,v,t}$	number of replicas of type v assigned to tier i at time t

TABLE 3: Optimization model decision variables.

5.3 Optimization Algorithm

The aim of this section is to provide a detailed description of the optimization algorithm implemented within SPACE4Cloud to solve (P). The faced optimization problem is challenging not only because of its $\mathcal{NP}$-hard nature but also because the assessment of a solution is time-consuming as it involves the evaluation of a set of $N = |\mathcal{T}|$ LQN models. For this reason a heuristic approach, capable of efficiently exploring the solution space, has been adopted. The rationale behind this approach consists in exploiting the two-level characterization of the problem, by using two different local search strategies (a process known as *hybridization*) to iteratively improve an initial solution. However, one of the major drawbacks of pure local searches is the possible cyclical generation of the same solutions, which is particularly detrimental when dealing with an time-demanding evaluation process. To prevent such undesirable behavior, a variant of the local search paradigm, known as Tabu Search (TS) [34], has been implemented, which exploits a memory structure (*tabu list*) to avoid generating recently assessed solutions.

The core of the optimization algorithm can be described as the alternation of two main phases, each one dedicated to the optimization of one level of the problem (see Algorithm 1): the upper level (selection of the must suitable resource/service type per tier) is addressed by the *TSMove* sub-routine whereas the lower level is tackled by the *ScaleLS* procedure. In this hybrid solution, *TSMove* implements a *tabu search* scheme whereas *ScaleLS* realizes a GRASP [35] technique, independently applied on all the intervals of the reference time horizon. The algorithm also implements a *Restart* mechanism that re-runs the optimization process starting from a new solution generated to address the search toward poorly explored zones of the solution space. For this reason, to avoid loosing good solutions, an elitist mechanism is also present, meaning that the algorithm keeps track of the best feasible solution discovered along the overall search: the *Best*. Nevertheless, the algorithm manages also the best solution found between two restarts, namely the *LocalBest*.

As far as the tabu paradigm is concerned, this is implemented by means of two memory structures: a *short term* memory and a *long term* one. The short term memory stores hash-coded information about the most recently generated solutions by *TSMove* and its goal is to avoid cycles in the tabu search phase of the algorithm. The long term memory, in turn, is used to store the frequency of assignments and evaluations for a particular resource type and tier. The main difference with respect to the short term memory is that this structure uses only the assignment of resource types to tiers to build the hash key, while the short term memory uses information about the entire solution. The goal of this memory is to implement an *aspiration criterion* (more details can be found in [42]) that enables the search process to escape from local optima, breaking free from constraints imposed by the short term memory; through this structure the algorithm is able to quickly identify poorly explored regions of the search space for the *type assignment* problem (i.e., the assignment of VM types).

Algorithm 1 sketches the structure of the proposed optimization method. The algorithm starts with the initial solution (step 1), obtained by the *Initial Solution Builder* solving the MILP problem presented and discussed in [17]. Since the *ScaleLS* procedure operates on feasible solutions only, if the initial solution is infeasible, the *MakeFeasible* procedure (step 15) is executed. This procedure acts progressively increasing the number of virtual resources of each tier until a feasible configuration is found. Once feasibility is achieved, *ScaleLS* is performed (at step 16). The goal of this procedure is to find the minimum number of replicas for each tier to fulfill the QoS requirements. It implements the GRASP that operates independently, and in parallel, on all of the

Algorithm 1: Optimization Algorithm

```
Input : $MaxIter$                    // Maximum number of iterations
Output: $Best$                       // Optimized solution
1  $LocalBest \leftarrow$ MILP()
2  $Best, Current \leftarrow LocalBest$
3  $Iter \leftarrow 0$
4  $MemST, MemLT \leftarrow ()$
                    // Initialization of memory structures
5  while $Iter < MaxIter$ do
6      if $Iter > 0$ then
7          $Candidate \leftarrow$ TSMove($Current$, $MemST$)
                                  // New candidate solution
8          if $Candidate = Current$ then
9              $Candidate \leftarrow$ Restart($Current$, $MemLT$)
                                  // Aspiration mechanism
10             $LocalBest \leftarrow Current$
11         $Current \leftarrow Candidate$
12     UpdateMemST($MemST$,$Current$)
                         // Updating the short-term memory
13     UpdateMemLT($MemLT$,$Current$)
                          // Updating the long-term memory
14     if $Current$ is not Feasible then
15         $Current \leftarrow$ MakeFeasible($Current$)   // Repair action
16     $Current \leftarrow$ ScaleLS($Current$)            // Local search
17     if $Current < Best$ then       // Update the best solution
18         $Best \leftarrow Current$
19     if $Current < LocalBest$ then
20         $LocalBest \leftarrow Current$     // Update the local best
21     else
22         $Current \leftarrow LocalBest$
23     $Iter \leftarrow Iter + 1$
```

24 periods; it terminates when a further reduction in the number of replicas would leads to an unfeasible solution.

The solution space is explored at the upper-level (i.e., type assignment) by means of the *TSMove* procedure (step 7). This procedure selects randomly one of the tiers and alters the related resource type. The selection of a new type is based on *a roulette-wheel* (also known as *fitness proportionate*) mechanism [42]. The efficiency of the resource, represented by the speed/cost ratio ($f_v = \frac{S_v}{Avg[C_{v,t}]}$), has been used to estimate the fitness of the each resource type. The roulette selects a certain resource type with a probability proportional to its fitness f_v, that is $p_v = \frac{f_v}{\sum_{v \in \mathcal{V}} f_v}$; in this way, types that are more likely to generate high-quality solutions are preferred in the selection process. This selection method is commonly used in genetic algorithms but has been demonstrated to be beneficial also in other approaches.

The short term memory ($MemST$) is used by TSMove to avoid generating solutions with a combination tiers/resource types that has already been evaluated. However, at some point it may occur that the short term memory impedes the generation of any new solution; when such a situation arises a restart mechanism based on the long term memory ($MemLT$) is performed to build a new current solution from poorly visited areas of the search space (step 9). In order to produce a new configuration the algorithm retrieves the resource type that has been less frequently evaluated for each tier. Noteworthy is that every time an action affecting upper-level decisions (*TSmove* or *Restart*) is executed, it alters the structure of the current solution so much that the lower-level optimization gets invalidated. Consequently, the *MakeFeasible* and *ScaleLS* procedures must be performed on the newly generated solution. More details can be found in [43].

Note that, if QoS or service assignment constraints are too stringent and a feasible solution cannot be identified, SPACE4Cloud returns the best unfeasible solution found, that is the one that violates the minimum number of constraints.

A brief analysis of the complexity of the optimization algorithm is provided at the end of this section. As is clear from the pseudo-code presented, the algorithm consists of a core loop repeated MaxIter times. The main operations performed in the cycle are, i) TSMove which generates a new solution in constant time, ii) ScaleLS and iii) MakeFeasible. Assessing the complexity of the latter two procedures is not immediate however if we indicate with Q the maximum number of VMs assigned to a tier (considering all tiers and all instants of time), it is possible to upper-bounding the number of operations performed by each of the two procedures. In fact, considering that both methods can not increase or reduce the number of resources per tier more that Q times in each time period their complexity is bounded by $\mathcal{O}(N \times |\mathcal{I}| \times Q)$. Consequently, the overall algorithm's complexity is $\mathcal{O}(MaxIter \times N \times |\mathcal{I}| \times Q)$. It is, nevertheless, useful to remark that the complexity of the algorithm is negligible compared to that of LINE [29] (the state-of-the-art LQN solver used in this study), which usually features a compound convergence time that it one order of magnitude larger.

6 Experimental Analysis

To evaluate the effectiveness of the presented approach, two main experimental campaigns have been carried out. In the first one, described in Section 6.1, SPACE4Cloud is exploited in the case study introduced in Section 2 to analyze the characteristics of the system under design, identify its limitations and bottlenecks and enhance the deployment configuration. The second analysis, presented in Section 6.2, aims at evaluating the scalability of the proposed solution with respect to the problem size using a synthetic benchmark. In both campaigns, our solution has also been compared against a first principle heuristic currently used by practitioners and SPACE4Cloud outperformed the first principle heuristic in both experiments, being able to identify feasible cloud allocations that are significantly less expensive (5-20% for the case study and 40-60% for the synthetic benchmark). Finally, it is worth highlighting another significant experimental evidence, namely that our solution has proved capable of optimizing the deployment of applications with a large number of tiers in times compatible with an iterative design process (order of minutes on commodity machines).

6.1 Case Study Analysis

With the purpose of assessing the applicability of our approach in a real-world scenario, SPACE4Cloud has been used to study the architecture of the *Constellation* extension to Modelio presented in Section 2.

6.1.1 Initial analysis

The analysis of Constellation has been based on the architecture described in Section 3.1, which includes the definition of QoS constraints, service allocation constraints, and demanding times, and the expected user behavior defined in Section 3.2.

The initial analysis has been focused on SVN and HTTP services since they are perceived by Softeam as the most critical for the first release of Constellation. Also, they are the most used services according to the user behavior defined for the system (accounting for the 85% of the workload). For the purpose of this analysis, Microsoft Azure IaaS services have been considered as target resources.

To understand how *Constellation* would react to the increase of its users, various workloads have been considered in different executions of the optimization procedure. Specifically, the normalized bi-modal workload reported in Figure 6 (which has been obtained from the analysis of real logs) has been scaled in such a way that the peak values would range from 50 to 500 users, which Softeam considered, respectively, the worst and the best case scenarios for the first release of Constellation.

The main findings of this initial analysis are summarized hereunder. For workloads featuring peaks of users below 200, SPACE4Cloud proposes to exploit Azure medium-sized virtual machines to host the HTTP Agent, whereas it proposed to host the SVN Agent on the most powerful VM type available. Unfortunately, when workloads with higher user peaks are investigated, the search leads to unfeasible (the constraints associated to the *commit* and *update* operations offered by the SVN Agent are violated no matter which cloud configuration is considered) and to the identification of more expensive deployments (due to the selection of more costly VM types to mitigate the gap between expected performance and the constraints imposed).

The main reason behind these results is the fact that, according to the Service Allocation Constraints defined on the Constellation architecture model (see Section 3.1), the SVN Agent cannot be replicated; therefore, it cannot rely on the cloud elasticity to deal with increasing workloads. Furthermore, it exposes frequent and time demanding operations (namely, *commit* and *update*, see Table 1). SPACE4Cloud seeks to compensate the rigidity imposed by the allocation constraints by scaling out the other components of the architecture trying to improve the system performance; however, since the SVN agent is already allocated on the most powerful machine, as the load increases, it becomes the bottleneck and the system inevitably reaches the point where no feasible deployment can be found. The other components, instead, are less problematic as they can either scale or are used less frequently. Thus, this study has been particularly useful to uncover the SVN Agent as the weakest point and the bottleneck of the architecture.

This is the typical situation wherein a reiteration is needed. In the Constellation case, the results achieved by SPACE4Cloud have determined the decision to revise the underlying architectural decisions and to build a new solution, henceforth referred to as *Conference*.

6.1.2 Reiteration

Conference has been created by the Softeam design team as an attempt to overcome the scalability limit singled out by SPACE4Cloud. Having identified the SVN Agent as the main bottleneck of the architecture, the company decided to decouple SVN updates (reads) from commits (writes) as much as possible. For this purpose, the set of models upon which the users operate has been divided into independent parts (fragments) replicated in separate repositories, managed by a central master server. In this way, the size of commits and the probability of conflicts is reduced; ultimately, exploiting multiple repositories also means being able to perform faster readings as they involve smaller models.

Consequently, in Conference, update operations are envisioned as executed by multiple replicated agents (*ReadOnlyConferenceReplica*), while commits are carried out by a singleton, namely the *Conference Agent*. Figure 5 depicts the Conference architecture. Some Constellation components (introduced in Section 3.1) have been maintained whereas a Conference Agent has

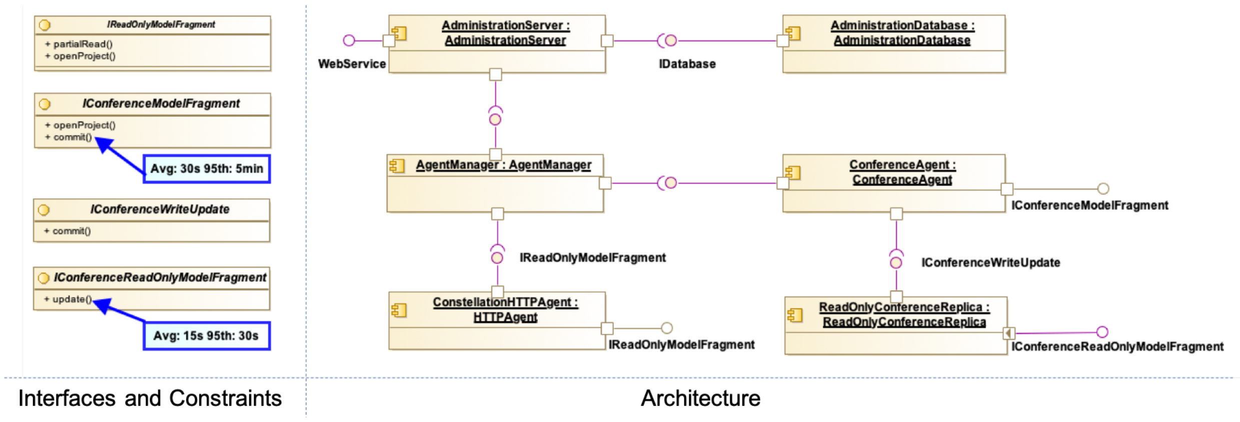

IReadOnlyModelFragment
+ partialRead()
+ openProject()
IConferenceModelFragment
+ openProject()
+ commit()
Avg: 30s 95th: 5min
IConferenceWriteUpdate
+ commit()
IConferenceReadOnlyModelFragment
+ update()
Avg: 15s 95th: 30s
AdministrationServer : AdministrationServer
WebService
IDatabase
AdministrationDatabase : AdministrationDatabase
AgentManager : AgentManager
ConferenceAgent : ConferenceAgent
IConferenceModelFragment
IReadOnlyModelFragment
IConferenceWriteUpdate
ConstellationHTTPAgent : HTTPAgent
IReadOnlyModelFragment
ReadOnlyConferenceReplica : ReadOnlyConferenceReplica
IConferenceReadOnlyModelFragment
Interfaces and Constraints
Architecture

Fig. 5: Conference service architecture and constraints

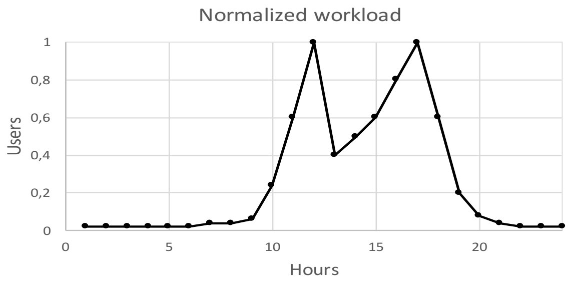


Fig. 6: Normalized workload for the Case Study analysis.

Operation	Average time (ms)
openProject (ConferenceAgent)	15,000
commit (ConferenceAgent)	1,500
update (ReadOnlyConferenceReplica)	1,000
commit (ReadOnlyConferenceReplica)	2,500

TABLE 4: Conference-specific demanding times.

Tier	50-200 Users	250-500
Admin Server	M	M
HTTP Agent	M	M
Conference Agent	M	L
ReadOnlyConferenceReplica	M	M

TABLE 5: Conference deployment at workload peak

replaced the central SVN Agent. As said, instead of executing large and sporadic commits (see Table 1 and Figure 3 for details), the Conference Agent commits small, yet more frequent, changes to the Master server, which, in turn, distributes the modifications to all replicas of the affected fragment. This is done by calling the *commit* method exposed by the ReadOnlyConferenceReplica, which host replicas of model fragments accessed by users via the *update* operation.

As for the previous analysis, some preliminary experiments have been performed to estimate the demanding times of the involved operations. Note that, since Conference was not implemented at the time the analysis has been carried out, we relied on an instance of the Constellation service reproducing a scenario in which small and frequent commits are performed instead of large and rare ones. The demanding times for the functionality implemented only in the Conference architecture are reported in Table 4 while demands associated to the Administration Server, Administration database and HTTP server are the same as those reported in Table 1. Notice that commit and update times are considerably shorter than those featured by the original architecture, due to the reduced size of the commits and parallel readings. Also, the expected user behavior changes with this architecture as users will update their local models either by updating directly from the central SVN agent or by doing partial updates and they will commit more frequently smaller chunks of data. The user behavior model has been updated consequently. More details can be found in [43].

Figure 7 shows a comparison between the expected response times for both the *commit* and *update* operations in the two considered architectures, *Constellation* and *Conference*. The graphs also show the threshold values set in the two cases as QoS constraints. From the comparison it results clearly that the *Conference* architecture is able to keep the response time below the threshold even when the maximum number of users grows up to 500. Figure 8 demonstrates a similar situation for what concerns the cost as the *Conference* system uses smaller-sized machines scaling the number of agents according to actual needs of capacity.

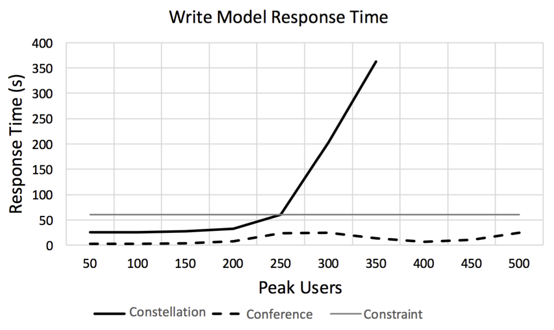
Table 5 reports the detail of the allocation of cloud services configurations at peak, obtained using the *Conference* architecture. In the table M, L, and XL refers to medium, large and extra large-sized Azure VMs, respectively.

6.1.3 *Comparison with best practices*

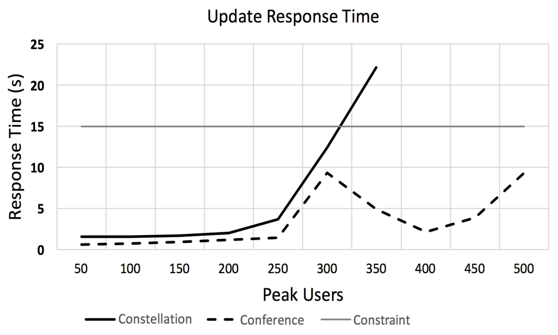
Finally, the quality of the deployment obtained via SPACE4Cloud has been compared against a best practice heuristic approach [44], [45], [38] currently implemented by IaaS providers to help customers to set suitable auto-scaling policies. The rationale behind this approach is to determine for a particular application a suitable deployment, that is, the number of cloud resources per tier, by imposing that the expected average CPU utilization of such resources must remain below a given threshold value.

This threshold-based heuristic has been implemented with the following two steps:

1) To reduce costs, for each tier, the cheapest resource type available at the cloud provider has been selected that satisfy the service allocation constraints defined (e.g., a requirement on the minimum memory) in the model.
2) To identify the number of needed replicas, two target CPU utilization values, of 60% and 80%, respectively



(a) Expected response time of the *commit* functionality in the *Constellation* and *Conference* architectures



(b) Expected response time of the *update* functionality in the *Constellation* and *Conference* architectures

Fig. 7: Analysis of the case study *Constellation* and *Conference* architectures

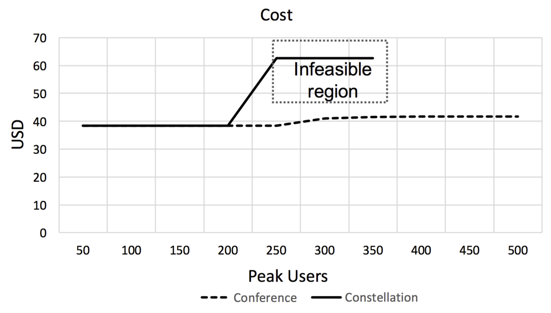


Fig. 8: Cost analysis of the two architectures under evaluation

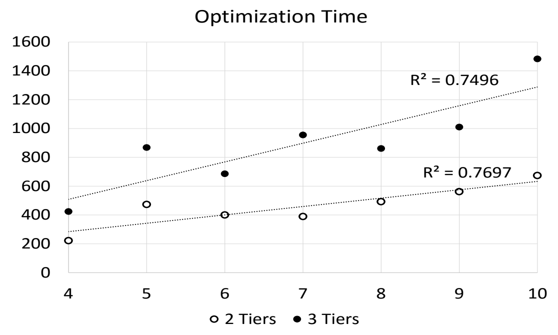


Fig. 9: Scalability Analysis

have been considered (as in [44], [45], [46], [47]). For a fair comparison, the CPU utilization for each hour of the considered time horizon is calculated using LQN models generated by SPACE4Cloud (see Section 4), which are eventually evaluated using the LINE solver.

Table 6 summarizes the outcome of the analysis conducted assessing the behavior of Conference under different workload conditions. In particular, Conference has been optimized for the bi-modal workload profile presented in Figure 6 scaled in such a way that the peak values range between 50 and 500 users. The table reports the deployment costs over the 24 hours obtained using SPACE4Clouds (for brevity called S4C in Table 6) and the presented heuristic considering 60% and 80% as utilization thresholds (H_{60} and H_{80}, respectively). Furthermore, the percentage deviations of SPACE4Cloud solution costs with respect to H_{60} and H_{80} outcomes, namely, Δ_{60} and Δ_{80}, are calculated and reported. SPACE4Cloud returns for all the experiments a cheaper solution; in particular, the percentage cost reduction ranges between 10% and 21% if we compare SPACE4Cloud and the most conservative heuristic (H_{60}) and within 3-6% when SPACE4Cloud and H_{80} are compared.

Users	S4C ($)	H_{60} ($)	H_{80} ($)	$\Delta_{H_{60}}$ (%)	$\Delta_{H_{80}}$ (%)
50	36.5	40.3	39.2	10.5	2.8
100	37.4	42.8	41.0	14.6	4.5
200	37.5	44.1	41.7	17.6	5.8
300	37.5	44.9	42.0	19.6	6.9
400	37.5	45.5	42.4	21.1	7.3
500	37.6	45.6	42.9	21.3	6.4

TABLE 6: Cost of the best solution found by SPACE4Cloud and by using a threshold based heuristic.

6.2 Scalability and Cost Analyses

This section analyzes how the proposed approach scales with respect to the problem instance size. In particular, the intention is to show that the presented approach is able to find an optimized allocation for complex architectural models within reasonable time and to investigate the behavior of the algorithm (e.g., which is the effect of the randomness on the solution space exploration? How does the best solution change during a search?). Finally, our solution is compared to the first-principle utilization threshold based heuristics considered in the previous section.

6.2.1 Experimental set-up

The architectural design problem faced in this work presents many dimensions that affect the solution cost and the time needed to derive the final optimized allocation. The main dimensions we have identified are: *i)* the *number of tiers* in the architecture under analysis as this defines the size of the design space in the formulation of the upper-level problem (see Section 5.2) as well as the complexity of the optimization algorithm (see Section 5.3), and *ii)* the *number of components*, which increases the complexity of the underlying LQN performance model. In order to test SPACE4Cloud performance, an extensive set of randomly generated application architectures, yet representative of real-world scenarios, has been analyzed. Since most of the real applications adopt two or three tiers ([48], [49]), the analysis focused on such cases whereas the number of components ranges from a minimum of 4 to a maximum of 10 distinct functionalities (as in [1], [50]). The execution of a request for each functionality involves some computation in components hosted on different tiers. Finally, Amazon EC2 has been selected as a target cloud provider.

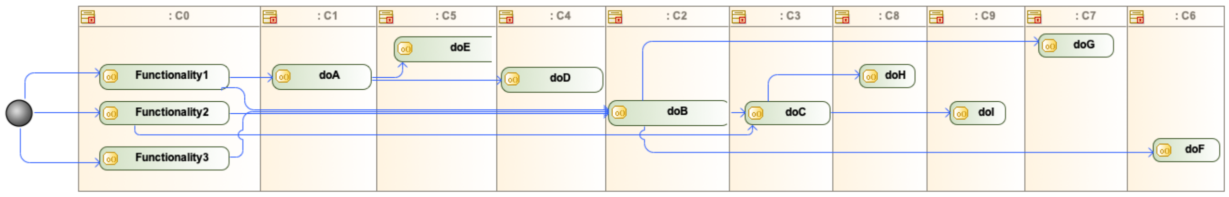


Fig. 10: Scalability analysis case study: distribution of calls within the system components.

Figure 10 refers to the largest model used in the analysis and highlights the interaction between the various components for the accomplishment of the three considered classes of functionalities. As an example, *Functionality 1* is offered by component 0, which is the entry point of the system, and generates a request to components 1 and 2. In order to serve the incoming requests, component 1 invokes functionalities offered by component 4 and 5. Functionalities 2 and 3 feature a similar behavior involving in the computation up to 10 components in total. These components are placed into three tiers; the first hosting components 0, 1, 4 and 9; the second hosting components 2, 6 and 5; and the third the remaining ones, that is 3, 8 and 7.

The components are assumed to be characterized by the same demanding times. This assumption corresponds to the worst case scenario for our tool. In fact, it corresponds to the case where all the tiers influence equally the overall performance and cost. Consequently, every tier has to be analyzed and optimized by SPACE4Cloud, which otherwise would have identified and mainly focused on the most critical ones. Ultimately, this means that numerous candidate solutions have to be evaluated. Vice versa, if the system has a single bottleneck, our algorithm focuses the analysis only on the corresponding tier, and the optimal solution can be identified more easily. The demanding times have been generated randomly according to other literature proposals [38], [10], [51], [47]. No memory constraint has been introduced to let the algorithm decide the type of VM among all possible types (thus exploring a larger solution space). Response time constraints have been set 10 times the value of the sum of the demanding times as in [40].

The analysis has been conducted on different versions of this application obtained by reducing the number of tiers from 3 to 2 or the number of components from 10 to 4 with the aim of studying how the time needed to find the optimized solution and the related cost changes with the complexity of the application model. Furthermore, for the most complex model (with 10 components and 3 tiers), the QoS constraints have been made progressively stricter up to the point that no feasible solution could be found. In this way, it was possible to verify that the optimization time of SPACE4Cloud does not increase significantly.

In all the experiments SPACE4Cloud has been executed on Amazon EC2, using a c3.xlarge virtual machine with four virtual CPUs and 7.5GB of memory.

6.2.2 *Results*

As in Section 6.1.3, here are reported and discussed the outcomes of a comparative experiment involving SPACE4Cloud and two utilization heuristics (denoted by H_{80} and H_{60}, respectively). This analysis considers the whole benchmark of models discussed in the previous section.

As shown in Table 7, SPACE4Cloud has always been capable of finding cheaper solutions compared to the utilization heuristics (confirming the results obtained for the case study) for all the considered scenarios. In particular, the average cost reduction with respect to the most conservative heuristic (with the utilization threshold set at 60%) is within 50-70% of the total cost while the cost reduction when the heuristic threshold is raised to 80% is reduced to 40-60%. Also in this case, the savings are lower for H_{80} compared to H_{60}, yet they are much higher than those presented in Section 6.1.3. This is mainly due to the presence, in the Constellation case study, of non-scalable components (the SVN agent), which act as bottlenecks and must be allocated on powerful and costly machines in order to cope with heavy workloads; consequently, this reduces SPACE4Cloud's margin of optimization. It also worth noting that the savings increase with the number of tiers $|\mathcal{I}|$ and with the number of components involved, $|\mathcal{C}|$.

$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{C}\rvert$	S4C ($)	H_{60} ($)	H_{80} ($)	$\Delta_{H_{60}}$ (%)	$\Delta_{H_{80}}$ (%)
2	4	18.7	41.0	53.6	65.1	54.4
	5	22.0	48.8	63.6	65.4	54.9
	6	26.6	61.7	78.8	66.2	56.9
	7	28.8	68.8	89.3	67.7	58.1
	8	31.8	77.0	99.5	68.0	58.7
	9	36.6	89.0	114.4	68.0	58.9
	10	38.8	96.1	124.8	68.9	59.6
3	4	26.0	42.6	54.7	52.5	39.0
	5	30.1	50.0	64.9	53.6	39.8
	6	33.7	61.4	80.7	58.2	45.1
	7	33.3	68.8	91.0	63.4	51.6
	8	37.6	76.7	101.0	62.8	51.0
	9	39.1	88.0	116.6	66.5	55.6
	10	42.4	192.0	145.2	77.9	70.8

TABLE 7: Comparison of average costs and savings

Figure 9 reports the time, in seconds, required to find an optimized solution (including also the execution time of the MILP solver to identify the initial candidate solution) for applications deployed in an infrastructure with two and three tiers. In both cases, the optimization time increases almost linearly with the number of application components. Even for the most complex case (an application composed of 10 components), this time remains below 12 minutes for the case of two tiers and less than 30 minutes for the case of three tiers. Since the solution space for a certain scenario does not depend on the number of components (see Section 5.3) the growth in the optimization time is mainly attributable to the increased complexity of the underlying LQN models.

Noteworthy is that the algorithm implemented by SPACE4Cloud exploits some randomness in order to better explore the solution space; in particular when performing the *TSMove*, in step 7 of Algorithm 1, the tier to which the tool applies the change of the resource type is selected randomly. Furthermore, the roulette wheel selection mechanism used to select the new type and performed within the *TSMove* is non-deterministic. In order to assess the robustness of the algorithm, relating it to the embedded randomness, several executions of the optimization

$\|\mathcal{I}\|$	$\|\mathcal{C}\|$	Cost ($)	σ_{Cost}	Time (sec)	σ_{Time}
2	4	18.7	0.3	222	12
	5	22.0	0.2	473	34
	6	26.6	0.6	400	26
	7	28.8	0.4	389	20
	8	31.8	0.5	492	19
	9	36.6	0.2	562	36
	10	38.8	0.2	674	45
3	4	26.0	0.5	424	67
	5	30.1	1.5	868	136
	6	33.7	1.5	686	53
	7	33.3	0.6	955	28
	8	37.6	2.4	861	28
	9	39.1	0.6	1010	27
	10	42.4	0.7	1483	57

TABLE 8: Comparison of SPACE4Cloud average and standard deviation of costs and execution time

procedure have been realized varying the random seed. Table 8 shows the average and the standard deviation for the execution time of SPACE4Cloud and the cost of the final solution, each entry of the table has been obtained by performing 10 runs of the optimization procedure. Overall, 140 instances have been evaluated. In all the experiments the algorithm proved to be stable in terms of the execution time, with a standard deviation of the execution time of at most 15.8% of its average value and a cost standard deviation of at most 6% of its average value.

7 Related Work

This research lays in the area of Model-Driven Quality Prediction (MDQP). Over the last two decades, several approaches have been proposed to integrate the prediction techniques of non-functional properties into the software engineering process and, nowadays, several tools exist that are intended to provide feedback to designers in order to improve performance and reduce costs of software systems. The starting point of the MDQP process is a set of models that describe the software system using an established modeling language such as UML. The second phase of the MDQP process is the automated transformation of architecture-level models into predictive performance models like LQN or Markov Chains (see, e.g., [52], [53]). Meta-models supporting performance predictions are surveyed in [54], [55] and [2]. The final goal of this transformation process is to enable the evaluation of different design configuration and support manual or automated optimization of the architecture at design-time.

The approaches that are more related to the one presented in this paper are those that allow users to optimize the architecture of a software system at design time, also taking into account the constraints and characteristics of the underlying resources needed for the execution. In what follows, these approaches are categorized distinguishing them into rule-based and meta-heuristic (SPACE4Cloud belongs to this last category).

Rule-based approaches: The first class of approaches encode best practices and user experience into feedback rules that are applied as transformations to modify a initial design [56]. In general, rule-based approaches require a continuous interaction with the user. This implies the evaluation of a limited number of alternative configurations and, as a result, the quality of the final solution depends on the user experience.

An example of this class of tools is the QVTR^2 framework proposed in [57]. It adopts the Query/View/Transformation (QVT) language for model-to-model transformations and extends QVT to support feedback rules that can be defined on non-functional requirements. The framework evaluates the performance of a candidate architecture and applies the rules specified by the user in order to derive new configurations.

Xu et al. present in [58] a semi-automatic framework, called Performance Booster, to analyze and optimize design configurations in a semi-automatic way. Their approach starts from UML models annotated with performance information. In order to overcome the burden of specifying feedback rules, their solution includes a set of default rules that encode some well-known best practices. The framework exploits LQN as performance model in order to evaluate the QoS of a candidate design. Rules are then applied in order to find performance issues (e.g., bottlenecks) and modify the performance model accordingly.

Parsons et al. propose in [59] a framework for detecting performance anti-patterns. It operates at run-time by monitoring various performance metric of the system. It exploits data mining techniques to identify meaningful links in the monitoring data, which are then fed into a rule-based engine in order to check the presence of anti-patterns.

A design time approach to identify performance anti-patterns has been proposed by Cortellessa et al. in [60]. Their approach relies on model-to-model transformations to derive performance models that are then analyzed in order to find anti-patterns. Once an anti-pattern is found, the system proposes alternative solutions to fix it. The main difference between this approach and other anti-pattern detection solutions is that the proposed solution is independent of the modeling language used.

Meta-heuristics: The second class, which includes our approach, leverages high-level algorithms, often bio-inspired, to efficiently explore the design space in search of solutions that optimize particular quality metrics. In [61], Li et al. introduce the AQOSA toolkit, which makes use of advanced evolutionary multi-objective optimization algorithms (i.e., NSGA-II and SPEA2) fully integrated with modeling technologies and performance analysis techniques. The main drawback of these algorithms is the significant number of evaluations needed in order to converge to the Pareto front. For this reason, only simple performance models can be used to evaluate each solution.

A similar approach has been adopted by Aleti et al. in the ArcheOpterix framework [11]. That work adopts the Architecture Analysis & Design Language (AADL) to model the application. The authors propose a generic framework that exposes most of the functionality needed to analyze and optimize the application model. In their evaluation, the authors focused on an embedded system. They selected a genetic optimization engine in order to optimize data transmission reliability and communication overhead while searching for an assignment to application components in resource containers.

An approach that combines the automation provided by heuristic approaches and the knowledge embedded in feedback rules is PerOpteryx [1]. This framework is designed for the optimization of component-based software architectures and is based on quality performance prediction techniques and meta-models. It implements a genetic algorithm (namely, NSGA-II) that has been modified to include the evaluation of feedback rules when generating a new candidate solution. More recently, this hybrid approach has been expanded by combining the use of an analytic optimization techniques with evolutionary algorithm. This tool shares many similarities with SPACE4Cloud, especially in the modeling language used to define the application and in the use of an analytic model in conjunction with a heuristic optimization.

The main difference is that the search space explored by our tool has been especially tailored for the optimization of cloud services. Cloud specific properties as the elasticity can not be considered using PerOpteryx alone, which, unlike our solution, is also able to generate optimized aggregations of components to be allocated to the same resource. Another main difference lays in the optimization approach itself: SPACE4Cloud uses a single-solution heuristic that, in general, requires a number of evaluations significantly lower compared to the other; consequently, it is much faster (a run requires minutes instead of hours [14]). However, PerOpteryx has the advantage of returning a set of feasible solutions representing possible trade-offs of different objectives. This provides the designer with a broad set of alternatives from which to choose. To combine the advantages of both tools, we proposed a toolkit and a unified workflow; the interested reader is referred to [14] for more details.

Frey et al. [13] presented a combined sim-heuristic approach to solve the problem of migrating existing enterprise software to cloud platforms considering a combination of a specific cloud environment, deployment architecture, and run-time reconfiguration rules. The design space is explored by means of a multi-objective genetic algorithm. This work has various elements in common with the one presented in this paper, yet it shows many differences as well. In particular, the tool by Frey et al. aims at easing the migration of legacy enterprise systems to the cloud, while SPACE4Cloud helps the designer to devise a cloud-ready application. From the technical point of view, our approach is less general but much faster; suffice it to say that the largest instance optimized by SPACE4Cloud in less than thirty minutes (see Section 6.2) while the CDO simulator (used in [13]) requires “from a few minutes to several hours” to evaluate a single solution. Moreover, our solution explicitly considers both service allocation and QoS constraints during the search process, giving the possibility to express QoS constraints for many criteria in terms of both average values and percentiles. Finally, our approach is able to exploit the cloud elasticity handling a 24-hour time horizon and solving 24 correlated capacity planning problems; other approaches only seek for solutions sized based on workload peak value. More recently, [62] proposed a multi-objective optimization framework which investigates multiple alternative deployments for component based applications considering on-premise, cloud, and hybrid topologies. The framework is very similar to PerOpterix and share the same optimization approach which requires several hours in cluster deployments to provide the Pareto front.

Two of our previous works [17], [63] deal with problems similar to the one addressed in this paper. [17] introduces the MILP model exploited by the Initial Solution Builder (presented Section 4) to provide an initial solution to the problem. This model, compared to the general formulation presented in Section 5, approximates the response time of a cloud deployment using M/G/1 queue models, and does not include constraints on the percentiles. The generated configuration, therefore, only represents an initial solution with ample room for improvement. In addition, that work assesses the impact of the adoption of this initial solution on the performance of SPACE4Cloud and the quality of the final solution. In [63] we generalize the model presented in [17] for the scenario of the capacity allocation problem of an application operated across multiple clouds simultaneously. The proposed MILP model identifies an initial deployment consisting in, for each hour of the day and each cloud provider, the type and number of virtual machines to be allocated to each application tier as well as the fraction of the total workload to be routed to the particular cloud.

8 Conclusions

This paper presented an optimization approach and a tool for the architectural design of cloud-native applications (deployed on public IaaS) characterized by variable workloads and subject to the time-varying performance of cloud infrastructures. Unlike previous literature proposals, the solution presented here addresses the problem of identifying a proper time-varying resource allocation (along a reference day) instead of performing the capacity planning at peaks. The corresponding optimization problem is $\mathcal{NP}$-hard and it has been solved using a meta-heuristic procedure implementing a hybrid two-level local search paradigm, which proved to be effective even for large-sized problem instances for various scenarios of interest. Furthermore, a comparison with commonly adopted techniques has shown that SPACE4Cloud tool outperforms the best practices providing cost saving up to 60%. Finally, the use of SPACE4Cloud in the design of an industrial case study has demonstrated that the proposed approach is capable of helping improve the application architecture and to support the software architect to identify adequate architectural solutions keeping costs under control.

Future work will consider the optimization of mixed IaaS-PaaS as well as multi-cloud applications also considering the availability metric among QoS constraints. Moreover, the adoption of robust optimization techniques will be investigated to cope with the uncertainty in the workload prediction and to support the choice of long-term contracts with cloud providers (e.g., Amazon reserved instances).

Acknowledgments

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds). Cloud costs for the usage of EC2 and Azure have been covered thanks to *Amazon AWS in Education Research* and *Windows Azure Research Pass* grants.

References

[1] A. Koziolek, H. Koziolek, and R. Reussner, “PerOpteryx: Automated Application of Tactics in Multi-objective Software Architecture Optimization,” in *Proc. of QoSA 2011*, 2011, pp. 33–42.

[2] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, “Software architecture optimization methods: A systematic literature review,” *Software Engineering, IEEE Transactions on*, vol. 39, no. 5, pp. 658–683, 2013.

[3] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev, “Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures,” *IEEE Transactions on Software Engineering*, vol. 41, no. 2, pp. 157–175, 2015.

[4] M. Scheerer, A. Busch, and A. Koziolek, “Automatic evaluation of complex design decisions in component-based software architectures,” in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, ser. MEMOCODE ’17, 2017, pp. 67–76.

[5] S. Becker, H. Koziolek, and R. Reussner, “The Palladio component model for model-driven performance prediction,” *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[6] M. Tribastone, S. Gilmore, and J. Hillston, “Scalable differential analysis of process algebra models,” *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 205–219, 2012.

[7] S. Gilmore and J. Hillston, “The pepa workbench: A tool to support a process algebra-based approach to performance modelling,” in *Computer performance evaluation modelling techniques and tools*. Springer, 1994, pp. 353–368.

[8] A. Megahed, A. Nazeem, P. Yin, S. Tata, H. R. Motahari-Nezhad, and T. Nakamura, "Optimizing cloud solutioning design," *Future Generation Comp. Syst.*, vol. 91, pp. 86–95, 2019.

[9] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics." in *NSDI*, 2017.

[10] G. Casale and M. Tribastone, "Modelling exogenous variability in cloud deployments," *SIGMETRICS Performance. Evaluation Review*, vol. 40, no. 4, pp. 73–82, 2013.

[11] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya, "Archeopterix: An extendable tool for architecture optimization of aadl models," in *Proc. of MOMPES'09*, 2009, pp. 61–71.

[12] A. Koziolek, D. Ardagna, and R. Mirandola, "Hybrid multi-attribute qos optimization in component based software systems," *Journal of Systems and Software*, vol. 86, no. 10, pp. 2542 – 2558, 2013.

[13] S. Frey, F. Fittkau, and W. Hasselbring, "Search-based genetic optimization for deployment and reconfiguration of software in the cloud," in *Proc. of ICSE '13*, 2013, pp. 512–521.

[14] M. Ciavotta, D. Ardagna, and A. Koziolek, "Palladio optimization suite: Qos optimization for component-based cloud applications," *EAI Endorsed Trans. Cloud Systems*, vol. 2, no. 6, 2016.

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

[16] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D.Gmach, R. Gardner, T. Christian, and L. Cherkasova:, "1000 islands: An integrated approach to resource management for virtualized data centers," *J. of Cluster Computing*, vol. 12, no. 1, pp. 45–57, 2009.

[17] D. Ardagna, G. Gibilisco, M. Ciavotta, and A. Lavrentev, "A multi-model optimization framework for the model driven design of cloud applications," in *Proc. of SBSE 2014*, 2014, pp. 61–76.

[18] A. Evangelinou, M. Ciavotta, G. Kousiouris, and D. Ardagna, "A joint benchmark-analytic approach for design-time assessment of multi-cloud applications," *Procedia Computer Science*, vol. 68, pp. 67–77, 2015.

[19] A. Kopaneli, G. Kousiouris, G. E. Velez, A. Evangelinou, and T. Varvarigou, "A model driven approach for supporting the cloud target selection process," *Procedia Computer Science*, vol. 68, pp. 89 – 102, 2015.

[20] A. Evangelinou, M. Ciavotta, D. Ardagna, A. Kopaneli, G. Kousiouris, and T. A. Varvarigou, "Enterprise applications cloud rightsizing through a joint benchmarking and optimization approach," *Future Generation Comp. Syst.*, vol. 78, pp. 102–114, 2018.

[21] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proc. of CLOUD 2013*, 2013, pp. 887–894.

[22] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "Cloudmf: Applying mde to tame the complexity of managing multi-cloud applications," in *Proc. of UCC 2014*, 2014, pp. 269–277.

[23] E. D. Nitto, P. Matthews, D. Petcu, and A. Solberg, *Model-Driven Development and Operation of Multi-Cloud Applications*. Springer, 2017.

[24] D. Franceschelli, D. Ardagna, M. Ciavotta, and E. Di Nitto, "Space4cloud: a tool for system performance and costevaluation of cloud systems," in *Proc. of MultiCloud '13*, 2013.

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

[26] J. Rolia and K. Sevcik, "The method of layers," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 689–700, 1995.

[27] L. Zhang, X. Meng, S. Meng, and J. Tan, "K-scope: Online performance tracking for dynamic cloud applications," in *Proc. of ICAC 2013*, 2013.

[28] W. Wang, G. Casale, A. Kattepur, and M. K. Nambiar, "QMLE: A methodology for statistical inference of service demands from queueing data," *TOMPECS*, vol. 3, no. 4, pp. 17:1–17:28, 2018.

[29] J. Perez and G. Casale, "Assessing SLA Compliance from Palladio Component Models," in *Proc. of SYNASC 2013*, 2013, pp. 409–416.

[30] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, "Enhanced modeling and solution of layered queueing networks," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 148–161, 2009.

[31] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-based reliability prediction with the palladio component model," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1319–1339, 2012.

[32] J. Rolia, G. Casale, D. Krishnamurthy, S. Dawson, and S. Kraft, "Predictive modelling of sap erp applications: Challenges and solutions," in *Proc. of VALUETOOLS '09*, 2009, pp. 1–9.

[33] N. Ferry, A. Solberg, P. Jamshidi, R. Osman, W. Wang, S. Seycek, V. Gligor, R. Sucasa, and A. Abhervé, "Modaclouds evaluation report – final version," MODAClouds EU Project Deliverable, 2015.

[34] F. Glover, "Tabu search: part i," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[35] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[36] N. Ferry, F. Chauvel, H. Song, A. Rossini, M. Lushpenko, and A. Solberg, "Cloudmf: Model-driven management of multi-cloud applications," *ACM Trans. Internet Technol.*, vol. 18, no. 2, pp. 16:1–16:24, Jan. 2018.

[37] C. Canali and R. Lancellotti, "Exploiting ensemble techniques for automatic virtual machine clustering in cloud systems," *Automated Software Engineering*, vol. 21, no. 3, pp. 319–344, 2013.

[38] D. Ardagna, M. Ciavotta, and R. Lancellotti, "A Receding Horizon Approach for the Runtime Management of IaaS Cloud Systems," in *Proc. of SYNASC 2014*, 2014, pp. 445–452.

[39] W. Wang and G. Casale, "Evaluating weighted round robin load balancing for cloud web services," in *Proc. of SYNASC 2014*, 2014, pp. 393–400.

[40] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[41] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[42] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009, vol. 74.

[43] G. P. Gibilisco, "A methodology and a tool for qos-oriented design of multi-cloud applications," Ph.D. dissertation, Politecnico di Milano, 2015.

[44] A. Wolke and G. Meixner, "Twospot: A cloud platform for scaling out web applications dynamically," in *Towards a Service-Based Internet*. Springer, 2010, pp. 13–24.

[45] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. Mckee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova, "1000 islands: An integrated approach to resource management for virtualized data centers," *Cluster Computing*, vol. 12, no. 1, pp. 45–57, 2009.

[46] J. Almeida, V. Almeida, D. Ardagna, Í. Cunha, C. Francalanci, and M. Trubian, "Joint admission control and resource allocation in virtualized servers," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 344 – 362, 2010.

[47] D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero, "A hierarchical receding horizon algorithm for qos-driven control of multi-iaas applications," *IEEE Transactions on Cloud Computing*, 2018.

[48] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2–19, 2012.

[49] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 253–272, 2013.

[50] A. Martens, D. Ardagna, H. Koziolek, R. Mirandola, and R. Reussner, "A hybrid approach for multi-attribute qos optimisation in component based software systems," in *6th International Conference on the Quality of Software Architectures, QoSA 2010*, vol. 6093. Springer, 2010, pp. 84–101.

[51] G. Casale and M. Tribastone, "Fluid analysis of queueing in two-stage random environments," in *Proc. of QEST 2011*, 2011, pp. 21–30.

[52] M. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, and J. Merseguer, "Performance by unified model analysis (puma)," in *Proc. of WOSP 2005*, 2005, pp. 1–12.

[53] N. Huber, F. Brosig, S. Spinner, S. Kounev, and M. Bähr, "Model-based self-aware performance and resource management using the descartes modeling language," *IEEE Trans. Software Eng.*, vol. 43, no. 5, pp. 432–452, 2017.

[54] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295–310, 2004.

[55] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Performance Evaluation*, vol. 67, no. 8, pp. 634–658, 2010.

[56] C. Trubiani, A. Bran, A. van Hoorn, A. Avritzer, and H. Knoche, "Exploiting load testing and profiling for performance antipattern detection," *Information & Software Technology*, vol. 95, pp. 329–345, 2018.

[57] M. L. Drago, C. Ghezzi, and R. Mirandola, "A quality driven extension to the qvt-relations transformation language," *Computer Science - R&D*, vol. 27, no. 2, 2012.
[58] J. Xu, "Rule-based automatic software performance diagnosis and improvement," *Performance Evaluation*, vol. 69, no. 11, pp. 525–550, 2012.
[59] T. Parsons and J. Murphy, "Detecting performance antipatterns in component based enterprise systems," *Journal of Object Technology*, vol. 7, no. 3, pp. 55–91, 2008.
[60] V. Cortellessa, A. D. Marco, R. Eramo, A. Pierantonio, and C. Trubiani, "Approaching the model-driven generation of feedback to remove software performance flaws," in *Proc. of SEAA 2009*, 2009, pp. 162–169.
[61] R. Li, R. Etemaadi, M. T. M. Emmerich, and M. R. V. Chaudron, "An evolutionary multiobjective optimization approach to component-based software architecture design," in *Proc. of CEC 2011*, 2011, pp. 432–439.
[62] F. Willnecker and H. Krcmar, "Multi-objective optimization of deployment topologies for distributed applications," *ACM Trans. Internet Technol.*, vol. 18, no. 2, pp. 21:1–21:21, Jan. 2018.
[63] M. Ciavotta, D. Ardagna, and G. P. Gibilisco, "A mixed integer linear programming optimization approach for multi-cloud capacity allocation," *Journal of Systems and Software*, vol. 123, pp. 64 – 78, 2017.

Michele Ciavotta received the Ph.D. degree in automation and computer science from Roma Tre, Italy in 2008. He is researcher at the University of Milano-Bicocca since 2017. His research work focuses on modeling and optimization of highly constrained combinatorial problems mainly arising in the fields scheduling and resource management of distributed systems.

Giovanni Paolo Gibilisco received the Ph.D. degree in computer engineering in 2016 from Politecnico di Milano, from which he also graduated in December 2012. His research interests lay in the area of model driven design of cloud based applications and software architecture optimization.

Danilo Ardagna is Associate Professor at the Dipartimento di Elettronica Informazione and Bioingegneria at Politecnico di Milano. He received a Ph.D. degree in computer engineering in 2004 from Politecnico di Milano, from which he also graduated in December 2000.His work focuses on the design, prototype and evaluation of optimization algorithms for resource management and planning of cloud computing and big data systems.

Elisabetta Di Nitto is Full Professor at the Dipartimento di Elettronica, Informazione and Bioingegneria at Politecnico di Milano, where she also earned her Ph.D. in Computer Science. Her current research interests are mainly on software engineering, and in particular, on process support systems, service-centric applications, dynamic software architectures, and self-adaptive systems.

Marcos Aurélio Almeida da Silva holds a Ph. D. degree in Computer Science of the Paris 6th University. At the time of the paper preparation Dr. Almeida worked in SOFTEAM as research engineer in projects related to modeling Cloud and big data applications.